Universidad de Alcalá
Escuela Politécnica Superior

MÁSTER UNIVERSITARIO EN INGENIERÍA DE TELECOMUNICACIÓN

Trabajo Fin de Máster

LOCALIZACIÓN EN INTERIORES DE AERONAVES LIGERAS MEDIANTE TÉCNICAS BASADAS EN LÁSER

Autor: DIEGO JUARA CASERO

Tutor/es: SEBASTIÁN SÁNCHEZ PRIETO

2018

Universidad de Alcalá

Escuela Politécnica Superior

Máster Universitario en Ingeniería de Telecomunicación

Trabajo Fin de Máster

LOCALIZACIÓN EN INTERIORES DE AERONAVES LIGERAS MEDIANTE TÉCNICAS BASADAS EN LÁSER

Autor: Diego Juara Casero

Tutor: Sebastián Sánchez Prieto

TRIBUNAL:

Presidente: Felipe Espinosa Zapata

Vocal 1º: Bernardo Alarcos Alcázar

Vocal 2º: Sebastián Sánchez Prieto

FECHA: JULIO de 2018

A Lorenza Carrasco, por enseñarme tanto.
1937 – 2018

ÍNDICE GENERAL

I RESUMEN

II MEMORIA

1 INTRODUCCIÓN 9
1.1 Presentación 9
1.1.1 Aeronaves ligeras no tripuladas 9
1.1.2 El problema del SLAM 13
1.2 Objetivos y motivación 16
1.3 Estructura del documento 17
2 ARQUITECTURA DEL SISTEMA 19
2.1 Sensores 20
2.1.1 Sensor láser 20
2.1.2 Sensor MARG 21
2.2 Bus I2C 22
2.3 Adquisición de datos 22
2.4 Bus SPI 23
2.5 Procesado de datos 23
2.6 Canal WiFi 24
2.7 Algoritmo SLAM 24
3 HARDWARE 25
3.1 Sensores 25
3.1.1 Sensor láser de distancia: VL53L0X 25
3.1.2 Sensor MARG: GY-87 26
3.2 Microcontroladores 27
3.2.1 Arduino Nano 27
3.2.2 ESP 8266 27
3.3 Plataforma 28
4 SOFTWARE 31
4.1 Adquisición de los datos 31
4.1.1 Sensor láser de distancia: VL53L0X 31
4.1.2 Sensor MARG: GY-87 32
4.2 Procesado de datos 34
4.2.1 Open ESP RTOS 34
4.2.2 Recepción de los datos 36
4.2.3 Conexión a red WiFi 37
4.2.4 Algoritmos de procesado de datos 37
4.2.5 Envío de datos 44
5 ALGORITMO SLAM 47
5.1 Fundamento teórico 47
5.1.1 Formulación matemática 47
5.2 Métodos SLAM 48
5.3 Elección del algoritmo SLAM 49

5.4 Estructura del algoritmo 51
5.4.1 Inicialización 52
5.4.2 Obtención de datos 53
5.4.3 Referenciar datos a sistema 53
5.4.4 Estimación de la posición 54
5.4.5 Actualización del mapa 55
5.5 Datos de salida 57
6 RESULTADOS 59
6.1 Parámetros modificables 59
6.1.1 Ancho de función de probabilidad de obstáculo 59
6.1.2 Parámetro de ajuste del algoritmo 60
6.1.3 Varianza del filtro de partículas 62
6.2 Pruebas en escenarios 63
6.2.1 Escenario 1: Sencillo 63
6.2.2 Escenario 2: Medio 66
6.2.3 Escenario 3: Dinámico 69
6.2.4 Escenario 4: Complejo 72
6.2.5 Escenario 5: Secuestro 75
7 CONCLUSIONES Y LINEAS FUTURAS 79
7.1 Conclusiones 79
7.2 Líneas futuras 82

III BIBLIOGRAFÍA

BIBLIOGRAFÍA 87

ÍNDICE DE FIGURAS

Figura 1.1 UAV RQ 2B Pioneer, en misión de reconocmiento en Irak. 10
Figura 1.2 Yamaha RMAX. 11
Figura 1.3 DJI Mavic PRO. 12
Figura 1.4 Detección referencias a partir de diferentes tipos de datos. 15
Figura 2.1 Esquema general del sistema. 19
Figura 2.2 Principio de funcionamiento de un láser tiempo de vuelo. 21
Figura 2.3 Transacción de ocho bits I2C. 22
Figura 2.4 Bus de comunicaciones SPI con varios usuarios. 23
Figura 3.1 Láser tiempo de vuelo a utilizar, VL53L0X. 25
Figura 3.2 Microcontrolador Arduino Nano. 27
Figura 3.3 Microcontrolador ESP8266. 28
Figura 3.4 Plataforma impresa para el prototipo. 28
Figura 3.5 Plataforma implementada. 29
Figura 4.1 Captura del programa MotionCal para calibrar el sensor de campo magnético. 33
Figura 4.2 Diagrama de bloques de las tareas y sus relaciones en el ESP8266. 35
Figura 4.3 Estructura de la trama enviada en el bus SPI. 37
Figura 4.4 Datos proporcionados por el sensor VL53L0X para una distancia de 72 mm. 38
Figura 4.5 En azul los datos proporcionados por el sensor VL53L0X sin filtrar, en rojo, filtrados con Kalman. 40
Figura 4.6 Angulos de navegación, *roll* , Φ, *pitch*, θ, y *yaw*, Ψ. 41
Figura 4.7 Esquema del algoritmo de Madgwick para un sensor MARG. 43
Figura 4.8 Estructura de los datos de la trama enviada a través de WiFi. 44
Figura 5.1 Representación de una zona del mapa generado por el algoritmo cuando encuentra un obstáculo $(x = 0, y = 0)$. 50
Figura 5.2 Esquema de bloques del algoritmo implementado. 51
Figura 5.3 Diagrama que muestra los sensores láser y el sistema con sus ejes de referencia. 53

Figura 5.4 Demostración del algoritmo de Bresenham, donde la línea roja es la deseada y los cuadros amarillos el resultado. 56
Figura 5.5 Captura del mapa generada por el *script* de generación de resultados. 58
Figura 6.1 Mapa generado en un entorno de 120×95 *cm* con diferentes valores del parámetro ancho de función de probabilidad de obstáculos. 60
Figura 6.2 Detección de un obstáculo dinámico con diferentes valores de α. 61
Figura 6.3 Diferentes varianzas con las que se generan las partículas. 62
Figura 6.4 Escenario 1. 64
Figura 6.5 Reconocimiento del entorno en el escenario 1. 65
Figura 6.6 Localización del sistema en el escenario 1. 66
Figura 6.7 Escenario 2. 67
Figura 6.8 Reconocimiento del entorno en el escenario 2. 68
Figura 6.9 Localización del sistema en el escenario 2. 69
Figura 6.10 Mapa obtenido del escenario 3. 70
Figura 6.11 Obstáculos introducidos dinámicamente. 70
Figura 6.12 Detección y localización al añadir obstáculos dinámicos. 71
Figura 6.13 Escenario 4. 72
Figura 6.14 Reconocimiento del entorno en el escenario 4. 73
Figura 6.15 Localización del sistema en el escenario 4. 75
Figura 6.16 Dispersión de las partículas para el escenario 5. 75
Figura 6.17 Posición inicial del sistema en el escenario 5. 76
Figura 6.18 Posición final del sistema en el escenario 5. 76

ÍNDICE DE TABLAS

Tabla 3.1 Perfiles de funcionamiento del sensor VL53L0X. 26
Tabla 4.1 Prioridades de las tareas en Open ESP RTOS. 36
Tabla 4.2 Tiempos medidos en la tarea de recepción de datos por el bus SPI. 37
Tabla 4.3 Tiempos medidos en la tarea de aplicación del algoritmo de Kalman. 40
Tabla 4.4 Tiempos medidos en la tarea de aplicación del algoritmo de fusión de datos. 44
Tabla 4.5 Tiempos medidos en la tarea de envío de datos por WiFi. 45

Parte I

RESUMEN

RESUMEN

En este trabajo se ha desarrollado un sistema que realiza SLAM a partir de los datos proporcionados por ocho sensores láser de distancia y un sensor MARG. Para ello, se ha construido una plataforma móvil en la que se integran los sensores. Los datos láser se procesan con un filtro de Kalman y se obtiene la orientación del sistema mediante un filtro de Madgwick. Tras esto, los datos se envían al algoritmo SLAM para obtener un mapa del entorno y la localización del sistema. Se han llevado a cabo diferentes pruebas demostrando el correcto funcionamiento del sistema.

PALABRAS CLAVE

SLAM, drones, robótica, láser tiempo de vuelo.

ABSTRACT

An SLAM system has been implemented from eight laser ranging sensor and a MARG sensor. To achieve that, all sensor has been integrated on a mobile platform. A Kalman filter is applied to laser data and orientation is obtained using Madgwick Filter. Following this, data is sent to SLAM algorithm in order to obtain a map of the environment and localice the system. Several tests has been carried out proving the good performance of the system.

KEYWORDS

SLAM, drones, robotics, time of flight laser.

RESUMEN EXTENDIDO

Los drones se están convirtiendo en los últimos años en un elemento con multitud de aplicaciones en una gran cantidad de áreas. Desde las aplicaciones militares, hasta el ocio, los drones cubren tareas tan dispares como monitorización de zonas de cultivo, detección de fuegos en bosques, vigilancia, reconocimiento en zonas de catástrofes, etc.

Como la mayoría de los robots con cierta autonomía, los drones, o la persona que los opera, tienen que ser conscientes del entorno en el que se encuentran, así como de su posición en éste. Sin embargo, cuando no se tiene información del entorno, como un mapa, es necesario que el propio dron sea capaz de obtener el mapa y su propia localización. Al problema de obtener un mapa y localizarse sobre este simultáneamente se le conoce como el problema del SLAM.

El problema en cuestión lleva resuelto desde hace varias décadas, sin embargo, existen aún ciertos retos que superar en entornos complicados, o muy extensos. La multitud de técnicas existentes para abordar el problema del SLAM es muy variada, al igual que el tipo de datos que se puede utilizar para realizarlo, entre ellos podemos encontrar imágenes, sonar, láser, odometría, etc.

En este trabajo se ha desarrollado un sistema que es capaz de realizar SLAM con ocho sensores láser que miden la distancia entre ellos y un objeto, y un sensor MARG (*Magnetic, Angular Rate and Gravity*) que proporciona las medidas de aceleración, velocidad de giro y valor de campo magnético en los tres ejes, necesarias para obtener la orientación del sistema. Los sensores utilizados, así como los microcontroladores utilizados, son de bajo coste.

El sistema desarrollado puede diferenciarse en dos partes. Una primera parte de adquisición y procesado de los datos proporcionados por los sensores, integrado en una plataforma alimentada por una batería que permite que sea móvil. Una segunda parte de aplicación del algoritmo de SLAM y obtención de resultados, que se lleva a cabo en un ordenador personal.

Se han seleccionado los láser VL53L0X y el sensor MARG GY-87. Estos se conectan a un microcontrolador Arduino Nano mediante un bus I2C. El microcontrolador es el encargado de configurar los sensores para su correcto funcionamiento, el bus de comunicaciones I2C y obtener los datos de los sensores. El Arduino Nano envía los datos

obtenidos al microcontrolador encargado de realizar el procesado de los datos mediante un bus SPI.

Los datos procedentes de los sensores deben ser procesados ya que presentan ruido o requieren de alguna transformación para que sean útiles. El encargado de llevar esto a cabo es el microcontrolador ESP 8266. El procesado aplicado a los datos de los sensores láser es un filtro de Kalman, que reduce la varianza de las medidas, proporcionando unos valores más "suaves". El procesado aplicado a los datos proporcionados por el sensor MARG es un filtro de Madgwick, que permite obtener a partir de los datos de aceleración, velocidad de giro y campo magnético los tres ángulos de navegación, *roll*, *pitch* y *yaw*. Con esto, los datos estarían listos para su utilización en el algoritmo, por lo que son enviados a un ordenador personal mediante una red WiFi.

El algoritmo SLAM que se ha implementado se basa en un algoritmo de código abierto conocido como *tinyslam*. Éste ha sido modificado para que funcione correctamente con el tipo de datos que nuestra plataforma proporciona. El funcionamiento se basa en un filtro de partículas, que son generadas con diferente posición, y a partir de éstas, se obtiene la que mayor verosimilitud guarda con los datos obtenidos y almacenados previamente, y se toma como posición estimada.

Para analizar los resultados, se ha creado un *script* en *Python*, que representa el mapa que se va obteniendo, la posición del sistema y su orientación, en tiempo real. El sistema se analiza en cinco escenarios diferentes, de los que se proporcionan imágenes y vídeos del sistema en funcionamiento.

- El primer escenario es el más sencillo, al ser ortogonal. En él, el sistema obtiene el mapa y se localiza correctamente incluso tras movimientos bruscos.
- El segundo escenario es más complejo, con obstáculos por medio. Aún así el sistema funciona correctamente, pero con ciertos problemas al pasar por zonas estrechas.
- En el tercer escenario se comprueba como reacciona el sistema cuando se incluyen nuevos obstáculos en un mapa que ya ha sido obtenido, y el resultado es correcto.
- En el cuarto escenario se analiza el funcionamiento del sistema cuando no se tiene un entorno cerrado, con obstáculos por el medio, y el sistema es capaz de localizarse correctamente, aunque con ciertos problemas en las zonas estrechas, como ya se había visto en otro escenario.

- Finalmente, en el quinto escenario, se ha comprobado como responde el sistema al problema del secuestro, y tras ajustar los parámetros del algoritmo, es posible obtener un correcto funcionamiento, pero esto presenta ciertos problemas, al ser la varianza del filtro muy grande.

Se observa que contar únicamente con ocho datos láser por cada conjunto de datos limita la resolución del sistema. Como era de esperar, cuanto más complejo es el entorno (menos ortogonal) el sistema presenta más problemas para realizar SLAM correctamente, sin embargo, si el escenario presenta límites rectos y bien delimitados, el sistema funciona correctamente. En definitiva, se obtiene un sistema que cumple con los objetivos planteados, es capaz de localizarse y obtener un mapa en diferentes entornos.

Parte II

MEMORIA

1 INTRODUCCIÓN

1.1 PRESENTACIÓN

1.1.1 *Aeronaves ligeras no tripuladas*

Las aeronaves ligeras no tripuladas (en ingles RPAS, *Remotely Piloted Aircraft System*, o UAV, *Unmanned Aerial Vehicles*), conocidas comúnmente como drones, son aeronaves que no cuentan con tripulación. A pesar de esto, son capaces de llevar a cabo planes de vuelo. Las ventajas que este tipo de aeronaves proporcionan son el bajo coste con respecto a tripuladas, tamaño y peso reducido debido a no necesitar tripulación, mayor maniobrabilidad y precisión en las operaciones, reducción del riesgo humano y el tiempo de formación para operarlo es reducido, entre otras.

Aunque podemos llevar a cabo una clasificación mucho más exhaustiva, la siguiente categorización cubre una gran parte del abanico de posibilidades actuales en el mundo de los drones:

- Tipo de Ala
  - Ala fija: Su vuelo es muy similar al de un avión ya que tienen las alas fijas.
  - Multirotor: Cuentan con tres, cuatro, seis o hasta ocho motores. Gracias a ello son capaces de mantener la posición durante el vuelo y maniobrar con mayor agilidad que los anteriores. Son los más extendidos en la actualidad.
- Autonomía
  - Controlado remotamente: Reciben directamente las instrucciones que deben realizarse en tiempo real enviadas por un operador en tierra.
  - Monitorizado: El vuelo del dron se realiza de forma autónoma, pero un operario es el encargado de tomar ciertas decisiones referentes al vuelo.
  - Autónomo: El dron es capaz de llevar a cabo un plan de vuelo preestablecido con total autonomía para tomar las decisiones en el momento en que sea necesario. Se ayuda de algoritmos que son capaces de decir la mejor acción para llevar a cabo la misión en función de los datos obtenidos por los sensores que montan.

Dentro del mundo de la robótica, los drones se han convertido en los últimos años en un elemento muy popular tanto para la industria y gobiernos como para el público general. Esto ha hecho que el mercado de los drones haya pasado de tener un tamaño muy reducido a ser un importante nicho de mercado. Es más, las previsiones de diferentes analistas de mercado coinciden claramente en que el tamaño de mercado de las aeronaves ligeras no tripuladas continuará creciendo en todos sus ámbitos de aplicación.

El mercado está comenzando a consolidarse, y prueba de ello es, entre otras, la gran cantidad de empresas que lo conforman, la fuerte especialización que están sufriendo éstas, desarrollando aplicaciones para tareas muy específicas de un sector, incorporando la inteligencia artificial y técnicas de *deep learning* o la formación de alianzas estratégicas entre empresas del sector.

La cantidad de aplicaciones para estos dispositivos ha crecido de manera exponencial. A continuación se comentan varias de las aplicaciones donde los drones tienen más impacto:

- Militar: Sin lugar a dudas es la aplicación con mayor cuota de mercado y en la que los vehículos aéreos no tripulados han llevado a cabo su evolución desde su nacimiento hasta nuestros días. Utilizados para el espionaje con misiones de reconocimiento u observación desde el aire. Permiten reducir el número de tropas efectivas en terrenos hostiles lo que garantiza una mayor seguridad para éstas, contar con una mayor información de la situación al efectuar una misión o, por ejemplo, alertar de posibles peligros de manera más eficaz. En la Figura 1.1 se puede ver el RQ 2B Pioneer.

Figura 1.1: UAV RQ 2B Pioneer, en misión de reconocmiento en Irak.

- Agricultura: Debido a su facilidad para sobrevolar campos completos en cortos periodos de tiempo, son ideales para el control

de plantaciones de manera remota, de hecho, en ciertos países como Japón lleva años utilizando el Yamaha RMAX (En la Figura 1.2. Éstos pueden recoger diferentes variables, como calidad del suelo, humedad y temperatura y de esta forma permitir mejorar la productividad, o fumigar las plantaciones. Se trata de una aplicación con gran mercado hoy en día. Y como planes de futuro, existen proyectos de ayuda a la polinización, con *drones abeja*, que debido a la reducción de la población mundial de abejas esto se había convertido en un problema.

Figura 1.2: Yamaha RMAX.

- Logística: ya hay proyectos de empresas para su uso en transporte de mercancías. En Rusia existen empresas utilizando drones para envío de pizzas. Sin embargo, en este ámbito se requiere una revisión de la normativa vigente y por ello aún no es algo común en el día a día en muchos países. Empresas como Amazon, DHL y Google se encuentran detrás de las iniciativas más esperanzadoras.

- Emergencias: Los usos en esta área son muy amplios debido a la facilidad de los drones de acceder a lugar sin importar las condiciones del terreno. Permiten identificar posibles rutas de acceso a lugares donde han ocurrido desastres naturales, evaluar las condiciones del lugar, trasladar material necesario, búsqueda de personas y/o animales, etc. Como ejemplo, tras la catástrofe de Fukushima, drones sobrevolaban la zona con el fin de monitorizar los niveles de radiación

- Seguridad: Gracias a las imágenes que pueden obtener desde las alturas, su uso en el ámbito de la seguridad es muy amplio. Son utilizados por la Guardia Civil en seguridad fronteriza, permiten realizar un control de los incendios forestales, etc.

- Ocio: Son muchas las personas para las que los drones se han convertido en un *hobby*, creándose un gran mercado y comunidad de estos dispositivos. En el mercado se puede encontrar desde configuraciones ya montadas hasta todas las piezas necesarias para montar tu propio drone. En la Figura 1.3 se puede ver el DJI Mavic PRO, uno de los más conocidos en el mercado de drones de ocio.

Figura 1.3: DJI Mavic PRO.

Para conseguir que el dron se sustente en el aire, trace las trayectorias deseadas y sea capaz de portar la carga útil, entre otras cosas, es necesario aplicar conocimientos de aerodinámica. La disposición de los motores y las hélices que ejercerán el empuje necesario, así como la capacidad de carga que pueden mantener, los materiales adecuados para los objetivos establecidos, deben ser estudiados mediante mecánica.

Además de los elementos necesarios para realizar el vuelo, el dron cuenta con multitud de sistemas, con el fin de recibir la información del entorno y poder llevar a cabo un vuelo correcto es necesario que cuente con sistemas que le doten de "sentidos", por ello se incorporan a él multitud de sensores, como cámaras. También es necesario la capacidad de comunicación, ya sea con un satélite o con tierra, para enviar datos de telemetría e información de la misión. Con el fin de llevar a cabo la misión, la carga útil deberá realizar unas determinadas tareas. Todos estos sistemas implican electrónica, comunicaciones y software, entre otras.

El ordenador de abordo es el cerebro del dron y tiene que realizar todas las operaciones, desde los algoritmos de control para mantener estable el dron y poder controlarlo, hasta activar la captura de datos

de ciertos sensores, pasando por la transmisión de telemetría por radio, entre otras cosas. Todo esto implica una fuerte participación de la ingeniería del software y sistemas empotrados.

Por tanto, los drones suponen un hito en robótica, uniendo en un mismo sistema tecnologías pertenecientes a diferentes ramas de la ingeniería. Este resultado proviene del desarrollo de muchos años, y es hoy en día cuando se comienza a ver con más claridad que los drones son una tecnología muy presente y que lo será aún más en el futuro.

1.1.2 *El problema del SLAM*

Uno de los problemas más presentes en robótica es determinar la posición y orientación del robot en el entorno en el que se encuentra. El robot, sea de la índole que sea (aéreo, terrestre, acuático, etc.) requiere conocer su posición referenciada al entorno en el que está operando, con el fin de poder tomar las decisiones adecuadas en función de éste. Por lo que el problema de la localización se basa en estimar la posición y orientación del robot en el mapa utilizando la información proporcionada por los sensores [1].

Como se ha comentado anteriormente, la localización es el proceso de posicionar y orientar el robot sobre un mapa del entorno en el que está operando. Sin embargo, esto implica proporcionar la posición del robot con respecto a alguna referencia del entorno en el que se está operando, pero si no se dispone de referencias del entorno, como por ejemplo, un mapa, ¿en referencia a qué se proporciona la posición y orientación del robot? La respuesta a esta pregunta puede parecer sencilla, la solución consiste en obtener el mapa del entorno a partir de los datos recolectados por los sensores de manera que integrando la información que nos proporcionan los sensores seamos capaces de obtener un mapa del entorno. No obstante, cabe destacar que al no conocer la localización, es difícil interpretar la información proporcionada por los sensores, y, a su vez, al no conocer el mapa, es complicado localizarnos en el entorno para saber dónde estamos obteniendo dichos datos.

Como solución a este problema, surgen las técnicas de SLAM (*Simultaneos Localization and Mapping*), localización y mapeado simultaneo. El objetivo de SLAM es obtener el mapa de un entorno desconocido *a priori*, a la vez que se realiza el posicionamiento del robot en ese mismo mapa que se está construyendo. Se reconoce como pioneros de esta técnica a Durrant-Whyte y John J. Leonard debido a su trabajo en [2], en el año 1991. Basándose en el trabajo previo de Smith, Self and Cheeseman en [3], investigaban la posibilidad de localizar un robot intentando emparejar referencias geométricas de un

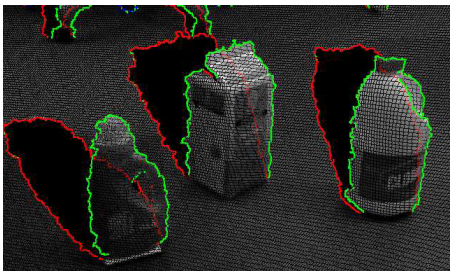
mapa *a priori* con los datos del entorno obtenidos a través de sensores sonar, junto con odometría, utilizando un filtro de Kalman extendido. SLAM es aplicable tanto en dos dimensiones como en tres.

El problema del SLAM, por tanto, consiste en interpretar la información que nos proporcionen los diferentes sensores con los que el robot cuenta, y realizar ciertas predicciones para obtener nuestra posición, así como un mapa del entorno por el que nos estamos moviendo. SLAM es más un procedimiento que una técnica determinada, puesto que para llevarlo a cabo se pueden usar diferentes métodos y, además, la investigación de nuevos métodos es un tema actual. Sin embargo, podemos definir unos sencillos pasos [4] que todo metodo de SLAM realiza:

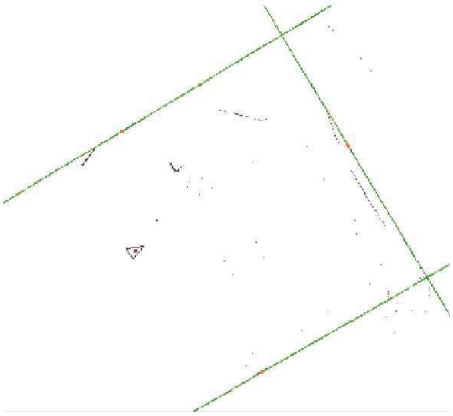
- **Obtención de referencias**: La información proporcionada por los sensores contiene referencias del entorno que utilizaremos para obtener nuestra posición. Estas referencias deben cumplir varios criterios, como por ejemplo, que sean detectables en las futuras adquisiciones de los sensores o que no cambien rápidamente a lo largo del tiempo. Este paso consiste en obtener esas referencias, y existen multitud de formas de llevarlo a cabo, como detección de líneas, de puntos aislados, de esquinas, etc. El hardware utilizado en este paso (los sensores) pueden ser múltiples, láser, sonar, cámaras, radar, etc. En la Figura 1.4 se pueden observar dos técnicas de extracción de referencias.

- **Asociación de referencias**: Consiste en relacionar las referencias que aparecen en la actual adquisición de datos por parte de los sensores, con las referencias vistas en anteriores adquisiciones de los sensores. Existen diferentes técnicas de llevar a cabo este proceso, como por ejemplo, utilizar la distancia euclidiana entre referencias actuales y posteriores para asociarlas.

- **Estimación y actualización del estado**: En esta fase se va a llevar a cabo la estimación del nuevo estado con respecto al anterior a partir de las referencias obtenidas en la actual adquisición de datos y la base de datos de referencias anteriores. Se trata del "cerebro" de todo el proceso, encargado de actualizar la posición del robot y el mapa que se está construyendo. Al igual que las otras fases, existen multitud de técnicas que podemos utilizar, lo más común en la bibliografía es encontrar técnicas que hacen uso del filtro de Kalman extendido (EKF, por sus siglas en inglés). Sin embargo, existen otras técnicas como por ejemplo estimación por nube de puntos. Una vez estimada la nueva posición del robot, se lleva a cabo la actualización del mapa y se incorporan a este los nuevos datos obtenidos por los sensores.



(a) Detección de bordes en imagen 3D.



(b) Detección de líneas a partir de datos sonar.

Figura 1.4: Detección referencias a partir de diferentes tipos de datos.

Como se ha comentado, existen multitud de métodos de SLAM y sensores aplicables a robots, basados en imágenes, odometría, odometría visual, uso de imágenes láser (2D y 3D), uso de sensores en el entorno, etc. Sin embargo, los requisitos a nivel de energía, peso, coste computacional, etc. son muy diferentes entre todos ellos. Si se pretende montar un sistema de SLAM en una aeronave ligera hay que tener en cuenta todos estos factores, ya que los recursos energéticos y de capacidad de carga serán bastante más limitados que en un robot terrestre. Es por ello que el sistema que el sistema (tanto hardware como software) que lleve a cabo el proceso del SLAM debe cumplir con los siguientes requisitos:

- Bajo peso: requieren que la carga que llevan sea la mínima posible, ya que las condiciones de estabilidad en vuelo son más complicadas que las de un robot terrestre. Además de esto también el peso influye enormemente en la autonomía del dron.
- Bajo consumo: es necesario que se maximice la autonomía del dron, por ello, es recomendable el uso de algoritmos sencillos, con baja carga computacional de forma que se optimice el consumo.

- Integrable: en relación a los requisitos anteriores, el espacio de carga en el dron es reducido, por ello, los sensores y dispositivos utilizados deben ser integrables.

1.2 OBJETIVOS Y MOTIVACIÓN

Como se ha comentado anteriormente, las técnicas de SLAM están muy desarrolladas en la actualidad. Sin embargo, muchas de estas técnicas en ocasiones requieren un hardware muy especializado, como LIDAR de altas prestaciones (como el HDL-32E).

En este trabajo se creará y analizará un prototipo de sistema para aeronaves ligeras no tripuladas que realice SLAM basado en sensores láser de tiempo de vuelo (junto con la ayuda de una unidad de medición inercial para conocer la orientación), utilizando componentes de bajo coste, al alcance del público general.

Se pretende que el sistema sea capaz de navegar por un entorno simulado con la capacidad de realizar un mapa de éste y obtener su propia localización únicamente con la información proporcionada por los sensores láser y la unidad de medición inercial.

No obstante, este trabajo no se centra únicamente en la implementación y correcto funcionamiento del algoritmo SLAM, sino que el sistema se creará desde cero, teniendo que llevar a cabo todas y cada una de las partes necesarias para su correcto funcionamiento.

Se lleva a cabo la creación de la plataforma, la integración de los sensores en esta, así como las conexiones eléctricas que sean necesarias. No se tienen en cuenta los requisitos de vuelo en cuanto a peso o dimensiones para hacer más fácil el prototipado. Con respecto al sistema de adquisición de datos, se creará un software para la lectura de los datos proporcionados por los sensores, que se realizará con la ayuda de un microcontrolador de bajo coste, así como el procesado necesario de estos datos. Una vez se tengan los datos aptos para el algoritmo SLAM, éstos serán enviados a un ordenador personal. El algoritmo de SLAM consistirá en una ligera modificación de uno de los algoritmos de SLAM con licencia libre que se pueden encontrar en la comunidad, y correrá en un ordenador personal para agilizar la integración de este con el sistema creado.

En definitiva, los objetivos principales de este trabajo quedan resumidos en los siguientes puntos:

- **Implementación de la plataforma para la adquisición de datos:** Integración de los sensores utilizados, configurarlos correctamente para obtener los datos con los requisitos necesarios para la aplicación.

- **Creación del código de procesado de los datos:** Crear el código encargado de procesar los datos obtenidos por los sensores para que estos sean útiles en el algoritmo.
- **Elección de un algoritmo SLAM libre:** Seleccionar un algoritmo SLAM de código abierto para utilizarlo como base en el sistema, y analizar detalladamente su funcionamiento.
- **Adaptación del código que realice el SLAM:** A partir del algoritmo usado como base, adaptarlo para hacerlo funcionar con los tipos de datos que proporcionan los sensores del sistema.
- **Realización de una aplicación de visualización de resultados:** Crear una aplicación que muestre en tiempo real los resultados obtenidos mientras el sistema está operando.
- **Evaluación del algoritmo de SLAM:** Evaluación de los diferentes parámetros que el algoritmo presenta y su impacto en los resultados. Comprobación del correcto funcionamiento del sistema en varios entornos simulados de diferente complejidad.

1.3 ESTRUCTURA DEL DOCUMENTO

En este documento se abordan los contenidos que se han desarrollado en este trabajo. Debido al hecho de que este trabajo consiste en el desarrollo de un algoritmo en una plataforma real, y que gran parte de este trabajo es la construcción y correcto funcionamiento del sistema de adquisición de datos, la parte práctica es bastante extensa. Sin embargo, el desarrollo teórico del algoritmo utilizado es imprescindible para el correcto funcionamiento del algoritmo SLAM.

La estructura del documento es la siguiente:

- **Capítulo 1: Introducción a las aeronaves ligeras no tripuladas y al problema del SLAM.** Introducción a las aeronaves ligeras no tripuladas y al problema del SLAM.
- **Capítulo 2: Arquitectura del sistema**. Se describe una idea general de la plataforma desarrollada, separando en bloques los diferentes subsistemas de los que consta y explicando el funcionamiento de cada uno de ellos.
- **Capítulo 3: Hardware.** Se proporciona la información de los elemenos hardware que se van a utilizar para realizar la implementación de cada uno de los bloques que constituyen el sistema.
- **Capítulo 4: Software**. Se detalla en profundidad el software desarrollado para que el hardware realice las funciones necesarias. Se desarrollan cada una de las operaciones llevadas a cabo en cada uno de los bloques para conseguir los objetivos.

- **Capítulo 5: Algoritmo SLAM.** Se introduce el algoritmo SLAM, su base teórica y diferentes métodos. Se detalla el algoritmo SLAM implementando, comentando previamente el algoritmo en el que esta basado, para posteriormente, analizar en profundidad cada una de las funciones que se llevan a cabo en éste.
- **Capítulo 6: Resultados.** Se evalúa el algoritmo implementado así como los diferentes parámetros modificables de los que éste dispone.
- **Capítulo 7: Conclusiones y líneas futuras.** Finalmente se desarrollan las conclusiones del trabajo realizado así como las posibles futuras actualizaciones al sistema.

2

ARQUITECTURA DEL SISTEMA

En este capítulo se detalla el sistema desarrollado desde el punto de vista funcional. Se comienza dando una idea general del sistema que se desea crear, y posteriormente se detalla cuál es el objetivo de cada uno de los bloques que forman el sistema.

Se pretende realizar un sistema que realice SLAM a partir de datos láser de distancia y datos procedentes de un sensor MARG (*Magnetic, Angular Rate and Gravity*). El sistema que se desea crear se diseña teniendo en mente la potencial aplicación en aeronaves ligeras no tripuladas en interiores, aunque en este trabajo únicamente se lleva a cabo un prototipo de este sistema.

La Figura 2.1 representa los bloques necesarios para que el sistema tenga todas las funcionalidades para poder cumplir los objetivos.

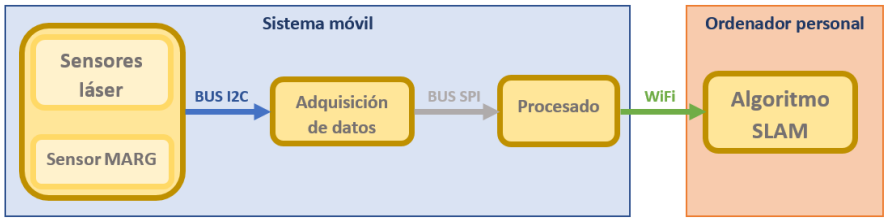


Figura 2.1: Esquema general del sistema.

Como descripción básica del sistema, se cuenta con unos sensores que obtienen los valores de las magnitudes deseadas. Los sensores láser miden la distancia a objetos cercanos, y el sensor MARG, proporciona los datos para obtener la actitud del sistema.

El sistema de adquisición se encarga de configurar el bus de comunicaciones I2C, los sensores, aplicar las calibraciones y obtener los datos. Posteriormente, los datos se transmiten al bloque de procesado a través del bus SPI.

Una vez se transmiten los datos por el bus SPI, el sistema de procesado se encarga de preparar todas las medidas para que los datos puedan ser usados en el algoritmo.

Con los datos procesados, éstos envían mediante una red WiFi, al ordenador personal encargado de realizar el algoritmo de SLAM, proporcionando las salidas necesarias para evaluar su rendimiento.

Como podemos ver, se hace uso de dos plataformas. Por un lado el sistema móvil que integrará los sensores, unidad de adquisición de datos y de procesado, que será alimentado por baterías, permitiendo que el equipo sea móvil para poder llevar a cabo las pruebas. Por otro lado, tenemos el ordenador personal que se hará cargo de realizar el algoritmo SLAM, que debido a su coste computacional es más difícil de integrar en un microcontrolador.

A continuación, en las siguientes secciones se detalla el funcionamiento de cada uno de los bloques del sistema.

2.1 SENSORES

Un dron requiere conocer información tanto del espacio que le rodea como de él mismo. Existen multitud de sensores que una aeronave ligera puede utilizar, sin embargo, en este trabajo nos centramos en incluir únicamente aquellos que son imprescindibles para el correcto funcionamiento del algoritmo SLAM, necesitando en una implementación real una mayor variedad de sensores.

En nuestro sistema, añadimos unos sensores láser de distancia con el fin de analizar qué se encuentra en los alrededores de éste, pero además, para llevar a cabo nuestro cometido de localización y obtención del mapa del entorno, es necesario conocer la orientación que tiene el dron con respecto a una referencia, por ello incluimos un sensor MARG. Estos elementos se detallan a continuación.

2.1.1 *Sensor láser*

El objetivo del sistema es ser capaz de obtener el mapa del entorno y posicionar el sistema en éste. Para lograr este objetivo, necesitamos obtener información del entorno, y esto lo realizaremos mediante sensores láser de distancia. Estos sensores nos proporcionarán la distancia existente entre nuestro sistema y el objeto que se presente delante del sensor. Esto es útil para el algoritmo de SLAM, ya que, gracias a estas medidas, puede ir integrando donde se encuentran los objetos en el mapa y si ese objeto ha sido visto previamente o no. Actúan de manera similar a como lo harían nuestros brazos, si entramos con los ojos vendados a una habitación que no conocemos. Seremos capaz de situarnos tras varias pasadas gracias a la información que nuestras manos nos proporcionarían y hacernos una idea de cómo es la habitación.

Estos sensores se conocen como sensores tiempo de vuelo (*Time of Flight*, ToF, por sus siglas en inglés), ya que ese es su principio de funcionamiento, como se puede ver en la Figura 2.2.

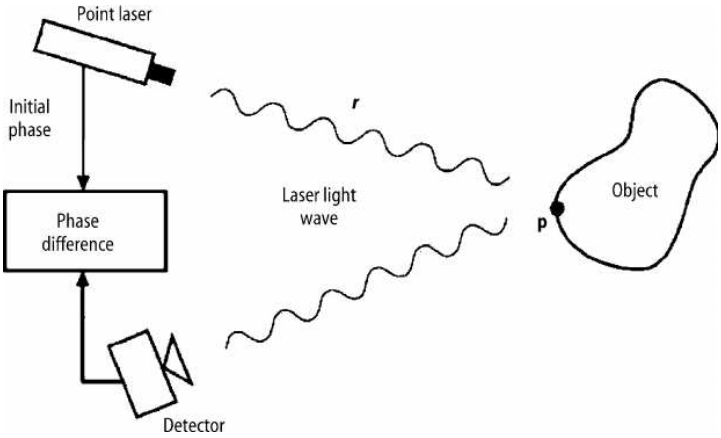


Figura 2.2: Principio de funcionamiento de un láser tiempo de vuelo.

Se calcula el tiempo desde que se emite la señal láser hasta que se recibe el eco de ésta, como se conoce la velocidad a la que viaja la señal, la velocidad de la luz, se puede obtener la distancia entre el sensor y el objeto en el que la señal ha sido reflejada siguiendo la ecuación 2.1.

$$d = \frac{t_{vuelo}}{2} c \tag{2.1}$$

Donde t_{vuelo} es el tiempo que tarda la señal en viajar y c es la velocidad de la luz.

Se utilizarán 8 láseres de tiempo de vuelo, colocados uniformemente cubriendo los 360°, por lo que se tendrá una separación angular entre ellos de 45°.

2.1.2 *Sensor MARG*

Únicamente con los sensores láser no somos capaces de saber si el sistema ha rotado, algo muy importante en la aplicación que se quiere conseguir. Para ello se incorpora un sensor MARG, que consta de una IMU (unidad de medición inercial) y un magnetómetro, para medir campo magnético. Este dispositivo proporciona valores de aceleración, velocidad de giro y campo magnético en los tres ejes, lo que permite, tras un procesado, obtener la orientación con respecto al norte magnético de la Tierra.

El acelerómetro y giróscopo son sensores microelectromecánicos (MEMS, por sus siglas en ingles), utilizan componentes mecánicos muy pequeños (del orden de micrómetros) capaces de convertir movimiento en electricidad (materiales piezoeléctricos) [5]. El funcionamiento del giróscopo consiste en una pequeña masa sustentada por

resortes que permiten que se mueva cuando el sensor se expone a giros, y estos movimientos son transformados en electricidad. El funcionamiento del acelerómetro consiste en algo similar. Se tiene una masa móvil en un único eje, sustentada por resortes. Cuando dicha masa se expone a una aceleración en ese eje, la masa se moverá y mediante diferentes técnicas, como por ejemplo, si esa masa móvil crea un condensador con una fija,es posible medir la aceleración a través del cambio de capacitancia. El funcionamiento del magnetómetro se basa en el efecto Hall. Este efecto se fundamenta en que si una corriente eléctrica es atravesada por un campo magnético perpendicularmente, se generará un campo eléctrico proporcional al valor del campo magnético, haciendo que a través del voltaje generado, sea posible cuantificar el campo magnético

2.2 BUS I2C

Para poder obtener los datos de los sensores es necesario que exista una conexión entre ellos y el bloque de adquisición de datos. Se ha seleccionado el bus de comunicaciones I2C.

Se trata de un bus de comunicaciones serie, que se basa en el modelo maestro-esclavo. La comunicación se lleva a cabo mediante dos líneas, SCL, encargada de transportar la señal de reloj con la que se sincronizan los datos, y la línea SDA, encargada de transportar los datos. El dispositivo que comienza la transmisión es el encargado de proporcionar la señal de reloj a la línea. Permite hasta 112 elementos en un mismo bus. A cada dispositivo se le asigna una dirección única en el bus, que es utilizada en el primer byte de cada transacción (enviado por le maestro) para identificar al destinatario de la transacción.

En la Figura 2.3 se puede ver una transacción de ocho bits en el bus I2C, donde vemos la condición de comienzo de transmisión al inicio, y la condición de parada al final.

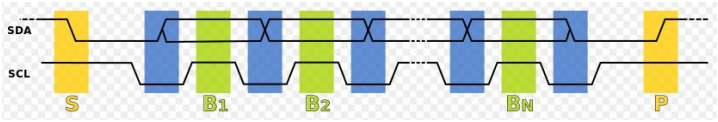


Figura 2.3: Transacción de ocho bits I2C.

2.3 ADQUISICIÓN DE DATOS

Antes de comenzar a adquirir los datos, es necesario configurar los sensores (modo de funcionamiento, calibración y frecuencia de actualización de datos) y el bus I2C (definir quién es el maestro y esclavo

y la velocidad a la que se quiere operar). También es necesario obtener los datos de los propios sensores, ya que estos los almacenan únicamente en sus propios registros. Este bloque es el encargado de llevar a cabo todas estas tareas, que correrá sobre un microcontrolador dedicado a la configuración y adquisición de los datos. Este microcontrolador contará también con una interfaz SPI para llevar a cabo la comunicación con el siguiente bloque.

2.4 BUS SPI

La comunicación entre el bloque de adquisición de datos y el siguiente bloque, se realiza mediante un bus SPI. Se trata de un estándar de comunicaciones serie. Cuenta con dos líneas de datos, salida del maestro entrada del esclavo (MOSI) y entrada del maestro salida del esclavo (MISO), una línea para la señal de reloj (CLK), y una señal de *chip enable* (CE). Es multiusuario (un maestro y varios esclavos), como se puede ver en la Figura 2.4, pero las comunicaciones son uno a uno.

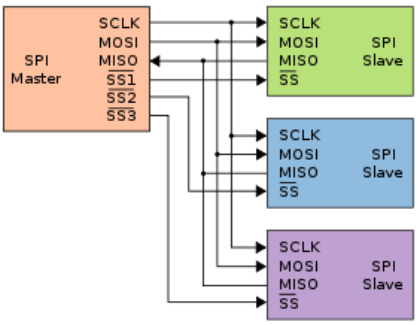


Figura 2.4: Bus de comunicaciones SPI con varios usuarios.

Cada partícipe de la comunicación cuenta con un registro de datos SPI (entre otros), donde se almacenan los datos para enviar (de byte en byte). Cuando se activa la señal de reloj, los datos que están en los registros de datos se intercambian con los del otro partícipe de la comunicación. De esta forma, una vez terminada la transacción SPI, en los registros de datos se encuentra el byte recibido.

2.5 PROCESADO DE DATOS

Los datos recibidos por el bloque de procesado no pueden ser utilizados directamente, ya que, contienen demasiado ruido, o no proporcionan información útil de forma inmediata, por ello, es necesario realizar un procesado de estos datos antes de que sean enviados al algoritmo SLAM.

Este procesado consiste en *"suavizar"* los datos proporcionados por los sensores láser de distancia, ya que los datos proporcionados por

éstos sufren una varianza que hace que no sean estables cuando la medida real debería serlo, y aplicar un algoritmo de fusión de datos a los proporcionados por el sensor MARG (aceleración, velocidad de giro y campo magnético) para obtener los tres ángulos de navegación útiles para nuestra aplicación, *roll*, *pitch* y *yaw*. En el capítulo correspondiente se explican en profundidad estos algoritmos.

Una vez que se tienen los datos válidos para su uso en el algoritmo SLAM, es necesario enviar estos datos al bloque de SLAM. Esto se hará mediante WiFi, por lo que el microcontrolador elegido debe permitir este tipo de comunicaciones.

2.6 CANAL WIFI

La comunicación entre el bloque de procesado y el del algoritmo SLAM se llevará a cabo mediante un red WiFi de 2,4GHz. Con el fin de no introducir retardos en la comunicación, se utilizarán paquetes UPD en los que se enviarán los ocho datos procesados del sensor láser y el ángulo de orientación *yaw*, ψ, obtenido del algoritmo de fusión de datos. Cada nuevo conjunto de datos procesado, se enviará un paquete UDP al algoritmo SLAM.

2.7 ALGORITMO SLAM

El algoritmo SLAM es el encargado de transformar los datos proporcionados por los sensores láser y el sensor MARG (ya procesados) en un mapa del entorno y la localización del sistema dentro de este. Existen multitud de técnicas para llevar a cabo este algoritmo, en función del tipo de datos que se estén recopilando del entorno. En nuestro sistema, se utilizará como base un algoritmo de código abierto acorde con el tipo de datos obtenido (datos láser y orientación), pero se realizarán las modificaciones necesarias para que funcione correctamente con nuestro sistema.

Debido a las exigencias computacionales del algoritmo SLAM, este debe correr en un ordenador personal. El algoritmo SLAM proporcionará los valores de posición actual del sistema, así como un mapa en formato gráfico del entorno en el que el sistema está operando. El funcionamiento interno del algoritmo SLAM se detalla de forma extensa en un capítulo posterior.

En el siguiente capítulo se detalla en profundidad el hardware elegido para construir el sistema.

3 HARDWARE

En este capítulo se detalla el hardware utilizado para el sistema.

3.1 SENSORES

El sistema obtiene información del entorno en el que se encuentra y de él mismo mediante los sensores. Se podría decir que los sensores dotan de sentidos al sistema para que pueda interactuar con el entorno y llevar a cabo su cometido de manera correcta.

3.1.1 *Sensor láser de distancia: VL53L0X*

El sensor láser seleccionado es el VL53L0X de la empresa ST, en la Figura 3.1. Se trata del sensor láser ToF más pequeño del mundo y ofrece precisión en las medidas independientemente del objeto en el que la señal se refleje. Las principales características se enumeran a continuación:

- Hasta dos metros de distancia.
- Precisión de 1 mm.
- Invisible al ojo humano.
- Inmunidad a la luz ambiente.
- Comunicación mediante I2C.

Este dispositivo tiene 3 modos de funcionamiento. El modo medida individual, en el que se tiene que pedir una muestra para que el sensor realice la medida y, tras finalizar, este vuelve al estado de reposo. El modo continuo, en el que el sensor está continuamente tomando muestras. Y el modo periódico, en el que el usuario define el periodo de tiempo que quiere entre medidas del sensor. Además de

Figura 3.1: Láser tiempo de vuelo a utilizar, VL53L0X.

estos modos de funcionamiento, el sensor cuenta con cuatro perfiles de medición que se detallan en la tabla 3.1.

Modo	Tiempo	Rango	Precisión
Por defecto	30 ms	1.2 m	± 5 %
Alta precisión	200 ms	1.2 m	± 3 %
Larga distancia	33 ms	2 m	± 5 %
Alta velocidad	20 ms	1.2 m	± 5 %

Tabla 3.1: Perfiles de funcionamiento del sensor VL53LoX.

3.1.2 *Sensor MARG: GY-87*

El módulo seleccionado, el GY-87, se compone de 3 sensores, un giróscopo, un acelerómetro y un magnetómetro. Esto es en realidad una IMU y un magnetómetro por cada eje. Los sensores son accesibles mediante un bus I2C. Cuenta también con un sensor de presión, pero no es usado en este trabajo.

El giróscopo proporciona un rango de medida de hasta $\pm$ 2000 $\frac{^\circ}{sec}$ y el acelerómetro de hasta $\pm$ 16 g con una precisión de 0,01 $\frac{^\circ}{s}$ y 60 μg, respectivamente. En cuanto a la frecuencia de salida de los datos, el giróscopo es configurable desde 4 Hz a 8 kHz y el acelerómetro de 4 Hz a 1 kHz. Este sensor cuenta con un procesador interno, llamado DMP, con un código propietario que calcula fusión de datos con 6 grados de libertad (datos del giróscopo y acelerómetro), proporciona cuaterniones, *roll*, *pitch* y *yaw*, detección de frame, etc. Sin embargo, no se va a hacer uso de éste, para tener un mayor control sobre el algoritmo de fusión de datos.

Este módulo cuenta también con un magnetómetro que permite medir el valor del campo magnético en los 3 ejes, con una precisión de 1 $^\circ$ a 2 $^\circ$. Su rango de funcionamiento es $\pm$ 8 G y cuenta con una precisión de $\pm$ 2 mG. Este sensor cuenta con dos modos de funcionamiento diferentes. El modo continuo, obtiene muestras a la frecuencia definida, y el modo medida única, en el que se ordena obtener una única muestra y se vuelve al estado de reposo. La frecuencia de salida de los datos puede llegar a ser de hasta 160 Hz en el modo más rápido.

3.2 MICROCONTROLADORES

3.2.1 *Arduino Nano*

El microcontrolador encargado de obtener los datos de los sensores será un Arduino Nano, en la Figura 3.2. Este microcontrolador es lo suficientemente potente como para llevar a cabo las tareas de adquisición de datos y, además, el desarrollo en Arduino nos permite utilizar la gran cantidad de bibliotecas que existen para esta plataforma gracias a su comunidad, a la vez que mantenemos las funciones más críticas en un microcontrolador independiente. Sus características principales son las siguientes:

- **Procesador:** ATmega328.
- **RAM:** 2 KB.
- **Memoria Flash:** 32 KB (2 KB bootloader).
- **Conexiones:** I2C, SPI, 22 pines digitales, 8 pines analógicos.

Figura 3.2: Microcontrolador Arduino Nano.

3.2.2 *ESP 8266*

El microcontrolador que se va a utilizar como cerebro del sistema es el ESP8266, se puede ver en la Figura 3.3. Se trata de un chip de bajo coste, que además cuenta con conectividad WiFi (pila TCP/IP completa). Entre sus características principales destacamos:

- **CPU:** Tensilica Xtensa LX106 a 80 MHz.
- **RAM:** instrucciones 64 KB, datos 96 KB.
- **Conectividad:** IEEE 802.11 b/g/n.
- **Conexiones:** 16 pines GPIO, SPI, I2C (software), DMA, UART y WiFi.

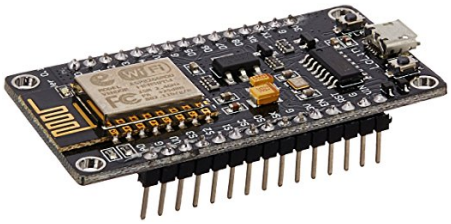


Figura 3.3: Microcontrolador ESP8266.

Existen multitud de entornos de desarrollo para este microcontrolador, Arduino, MicroPython, Lora, etc. Sin embargo, para la aplicación que se va a llevar a cabo se ha decido usar el entorno de Free RTOS adaptado a este microcontrolador, Open ESP RTOS. En la sección correspondiente se detalla en profundidad este *framework*.

3.3 PLATAFORMA

La plataforma sobre la que si situarán los sensores es un polígono octógono diseñado con la herramienta gratuita *Tinkercad* de *Autodesk*. Posteriormente es impreso en una impresora 3D con material PLA (ácido poliláctico). El resultado obtenido es el que se muestra en la Figura 3.4.

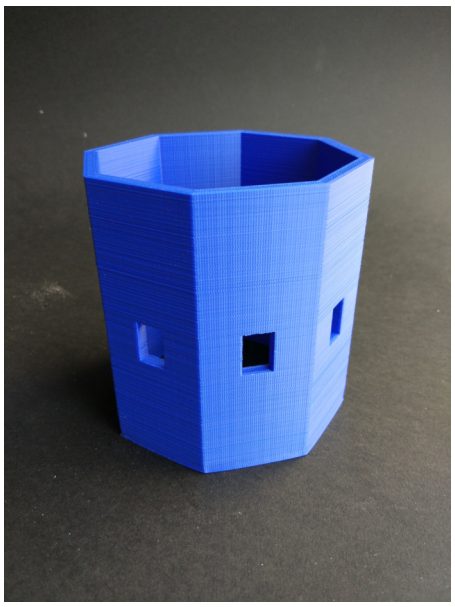
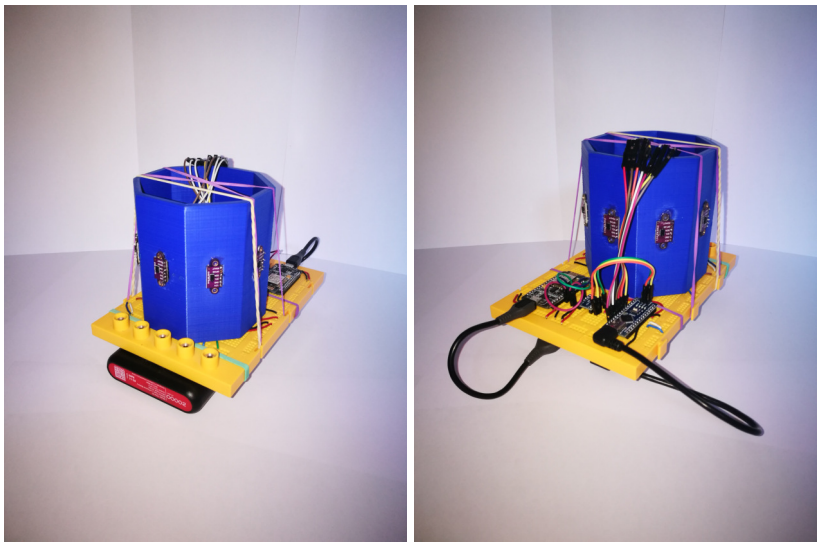


Figura 3.4: Plataforma impresa para el prototipo.

En los orificios se colocaran los sensores láser, de forma que las conexiones queden dentro de ésta. Todo se colocará sobre una *protoboard* para una depuración más rápida y accesible. Sobre la *protoboard* se colocará la IMU que proporcionará la actitud del sistema, el microcontrolador, el bus de comunicaciones I2C y alimentación. En la



(a) (b)

Figura 3.5: Plataforma implementada.

Figura 3.5 puede verse el sistema final. La batería portatil se encuentra debajo de la protoboard.

4 SOFTWARE

En este capítulo se profundiza en el software utilizado para hacer funcionar el sistema. Tanto el *framework* sobre el que se construye todo el sistema, los pasos adicionales para configurar el hardware de acuerdo a las necesidades y las tareas que los microprocesadores ejecutan. Todo el código desarrollado puede encontrarse en el siguiente repositorio.

https://github.com/djuara/tofslam

4.1 ADQUISICIÓN DE LOS DATOS

Los sensores se encuentran conectados a un bus de tipo I2C. Este bus está orquestado por el microcontrolador Arduino Nano, que actúa como maestro en la comunicación. El modo de funcionamiento es modo rápido, realizando las transacciones a 400 *kHz*. El microcontrolador es el encargado de configurar los sensores para trabajar en el modo que se desea así como de obtener los datos de estos. A continuación, se detalla la configuración para los dos tipos de sensores.

4.1.1 *Sensor láser de distancia: VL53LoX*

Para controlar el sensor mediante Arduino tenemos una biblioteca creada por Pololu a partir de la API oficial de ST. Esta biblioteca nos ofrece compatibilidad con Arduino y una mayor sencillez que la oficial, lo que para nuestro proyecto cumple con los requisitos.

Como se ha visto anteriormente, el funcionamiento del bus I2C se basa en direcciones, por lo que cada sensor tiene que tener su propia dirección. El sensor que se está utilizando tiene una dirección por defecto que siempre se autoasigna cuando el dispositivo se inicia, por lo que el primer paso a llevar a cabo cuando el software se inicializa es cambiar la dirección por defecto de estos sensores uno a uno para que no haya dos sensores con la misma dirección. De esta forma vamos activando los sensores uno a uno con su dirección asignada.

En nuestra aplicación utilizaremos el modo de lectura continuo. En cuanto al perfil de lectura de los datos, utilizaremos el modo de mayor rango, que permite que obtengamos muestras nuevas del sensor con una distancia máxima de 2 *metros* a una frecuencia de 33 *Hz*. Esta frecuencia es suficiente para la aplicación que estamos buscando, ya que nos proporciona treinta muestras por segundo del entorno

del robot, y el incremento del alcance de los sensores a dos metros mejorará el rendiemiento del algoritmo.

Este sensor no requiere ningún tipo de calibración externa en las condiciones en las que va a ser operado (él mismo realiza una serie de calibraciones internas), por lo que con esto el sensor láser queda configurado para su funcionamiento en el sistema.

4.1.2 *Sensor MARG: GY-87*

El sensor MARG se conecta de igual manera al bus I2C anterior. Para conectar con él se hace uso de las bibliotecas MPU6050 y HCM5883L, disponible en el paquete de bibliotecas `i2cdevlib`. Se trata de un paquete de bibliotecas para utilizar diferentes dispositivos a través del bus I2C. Es un proyecto de código abierto, bajo licencia MIT, dirigido por Jeff Rowberg.

La dirección por defecto del sensor MARG es diferente a las anteriores por lo que no hay conflicto de direcciones. Sin embargo, es necesario activar unas funcionalidades en la IMU (MPU6050) para poder obtener todos los datos necesarios, tanto del giróscopo, acelerómetro y magnetómetro.

Debido a que el sensor MARG tiene un bus I2C interno para acceder al magnetómetro, es necesario habilitar el acceso a este, mediante la funcion *bypass* para que el sensor que integra acelerómetro y giróscopo traslade las peticiones de datos del magnetómetro al bus interno donde éste se encuentra. Tras esto, es necesario también desactivar el modo *sleep*, para que el dispositivo no entre en modo bajo consumo.

En relación a la frecuencia de obtención de los datos del sensor MARG, es necesario definirlo en los 3 sensores que contiene, giróscopo, acelerómetro y magnetómetro. La frecuencia de obtención de datos del giróscopo y acelerómetro se ha seleccionado a 200 Hz, mientras que los valores del magnetómetro se actualizarán a 75 Hz, el máximo configurable en modo continuo. Estas frecuencias de actualización de los datos proporcionan unos valores adecuados para el algoritmo de fusión de datos, como se comentará más adelante.

Una vez hecho esto, el sensor MARG se inicializa. Sin embargo, los datos proporcionados por ésta no son aún válidos, pues es necesario calibrarlo.

Debido a defectos de fabricación, giróscopo y acelerómetro presentan un cierto *offset* en sus medidas, de forma que, si el sensor se encuentra situando sobre una superficie completamente horizontal, los valores que el giróscopo debería mostrar son $x = 0\frac{^\circ}{s}, y = 0\frac{^\circ}{s}$ y $z = 0\frac{^\circ}{s}$, ya que no hay movimiento, y para el acelerómetro $x = 0g, y = 0g$ y

$z = 1g$, sin embargo, esto no ocurre. El proceso consiste en ajustar estos *offset* hasta que obtengamos los valores esperados mostrados anteriormente. Estos valores de *offset* se aplican al MPU6050 que los almacena y corrige los datos antes de enviarlos por el bus I2C.

En cuanto al magnetómetro, el error que debemos calibrar es de distinta procedencia. Los errores que debemos compensar se pueden clasificar de dos formas, *hard-iron* y *soft-iron* [6]. Los errores *hard-iron* están relacionados con objetos fijos presentes en las inmediaciones del sensor que provocan un campo magnético, resultando en un *offset* de la medición del campo magnético terrestre. Como su efecto es un *offset*, basta con obtenerlo y eliminarlo de las medidas para corregirlo. Los errores *soft-iron* se deben a objetos que no tienen por que generar un campo magnético, pero que interfieren con él, distorsionando su valor en función del ángulo de medición. Como el campo magnético es el mismo en todas las direcciones (localmente), las medidas en todas las direcciones deberían dar lugar a una esfera (mismo valor implica mismo radio). Sin embargo, este error lo que provoca es una deformación de esa esfera en una elipse, pudiendo corregirse a través de una operación matricial de nueve coeficientes (matriz de 3x3). Para realizar la calibración se utiliza una aplicación desarrollada por Paul Stoffregen, llamada MotionCal. Mediante los datos del sensor en los tres ejes obtiene los valores de calibración necesarios, tanto el *offset* como los coeficientes necesarios para obtener la esfera en las mediciones del campo magnético. Este proceso se lleva a cabo con el sensor en su posición final en la plataforma, ya que, como se ha explicado anteriormente, el sensor es sensible al entorno que lo rodea. En la Figura 4.1 se puede ver la interfaz del programa de calibración.

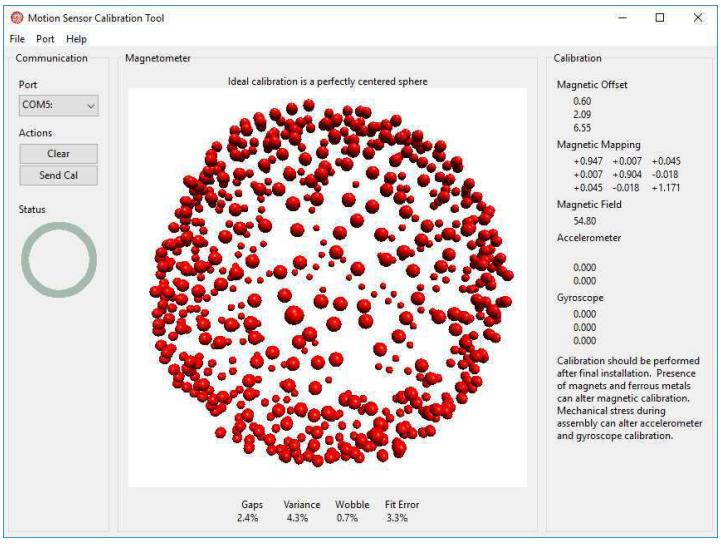


Figura 4.1: Captura del programa MotionCal para calibrar el sensor de campo magnético.

Una vez obtenidos los datos se elimina el *offset*, y se aplican las correciones para compensar los efectos *soft-iron* mediante la matriz de coeficientes obtenida por el programa a cada conjunto de datos que se obtiene del magnetómetro. Una vez aplicada la calibración, ese conjunto de datos es apto para su uso.

4.2 PROCESADO DE DATOS

4.2.1 *Open ESP RTOS*

Anteriormente se ha comentado que en el microcontrolador ESP 8266 se iba a utilizar un sistema operativo en tiempo real, pero, ¿qué es esto? Un sistema operativo en tiempo real es aquel que ofrece un comportamiento predecible en los tiempos de ejecución de las tareas que esta realizando [7]. Los sistemas operativos de carácter general no garantizan cuando llevarán a cabo una tarea o cuanto tiempo tardarán en realizarla, su comportamiento no es predecible, debido a la forma en la que comparte la CPU entre las distintas tareas, ya que su objetivo es garantizar una experiencia correcta al usuario, es decir, es más importante que el usuario no detecte que, por ejemplo, el ratón no responde correctamente, a que el navegador tarde 0,5 segundos más en abrirse.

En los drones, las exigencias referentes a los tiempos de respuesta de su controladora de vuelo son muy exigentes, ya que de otra manera el vuelo sería irrealizable o incontrolable. Además, el dron, aparte de volar, tiene que realizar otras funciones, como comunicarse, recibir datos de los sensores, procesarlos, etc. Sin embargo, estas otras tareas no deben interferir en las encargadas de controlar el vuelo. Esto hace que los sistemas operativos de tiempo real sean necesarios para los sistemas que controlan las acciones de los drones. Por ello, se ha seleccionado un sistema operativo en tiempo real como base para este trabajo, Open ESP RTOS.

Open ESP RTOS es una adaptación del sistema operativo para sistemas empotrados Free RTOS llevada a cabo por la comunidad (dirigido por la empresa Super House) única y exclusivamente para el dispositivo ESP 8266. Está basado en el SDK Espressif IOT RTOS, pero difiere de éste bastante.

Entre los componentes software libre tenemos FreeRTOS, el sistema operativo en tiempo real, lwIP, una muy conocida biblioteca ligera para comunicaciones TCP/IP en sistemas embebidos, y newlib, una biblioteca C de software libre para sistemas embebidos. El *framework* además contiene otros componentes, algunos no cuentan con licencia libre, como cierto código propietario de Espressif, del que únicamente se cuenta con los binarios.

Open ESP RTOS se trata de un *kernel* que proporciona las funcionalidades necesarias para desarrollar aplicaciones sobre él que cumplan con los requisitos de tiempo real. Además, como se ha comentado, incluye las funciones necesarias para realizar comunicaciones TCP/IP, necesarias en nuestro proyecto, como se ha comentado anteriormente.

A partir del *framework*, se crean las tareas necesarias que realice el microcontrolador, las cuales son:

- **Recepción de los datos:** Esta tarea será la encargada de comunicarse con la placa Arduino nano a través del bus SPI y obtener los datos de los sensores.
- **WiFi:** Esta tarea es la encargada de iniciar la conexión del módulo de comunicaciones WiFi del ESP 8266 con la red a la que nos vamos a conectar.
- **Procesado de los datos:** Una vez se obtienen los datos, es necesario procesarlos adecuadamente antes de enviarlos para ser utilizados por el algoritmo SLAM.
- **Envío de datos:** Debido a las capacidades del ESP 8266, el procesado del algoritmo SLAM debe llevarse a cabo en un ordenador personal, por ello es necesario enviar los datos mediante WiFi a este.

En la Figura 4.2, se puede ver un diagrama de bloques de la estructura de tareas que correrá sobre el sistema operativo en tiempo real.

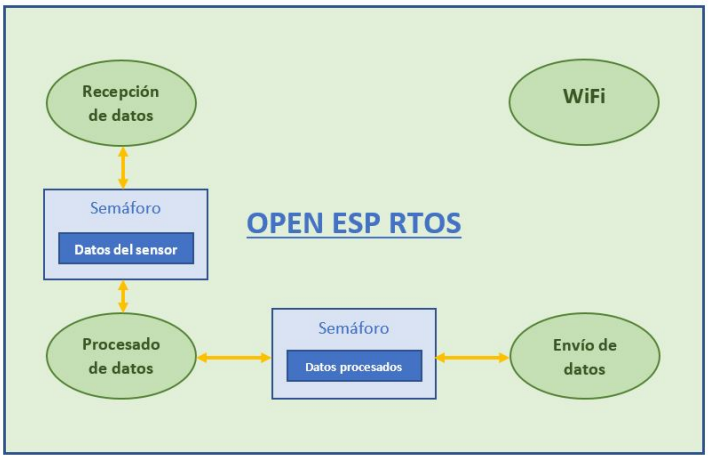


Figura 4.2: Diagrama de bloques de las tareas y sus relaciones en el ESP8266.

En ella se muestran las tareas que comparten datos entre ellas. Con el fin de garantizar el correcto funcionamiento, es necesario aplicar mecanismos de seguridad para cuando una tarea está leyendo unos

datos compartidos, éstos no sean modificados. Para ello se utilizan semáforos en las estructuras que son compartidas por varias tareas.

A partir de las tareas que se quieren implementar, es posible llevar a cabo una asignación de las prioridades que tendrán cada una de las tareas. La tarea más importante es la adquisición de los datos, ya que sin ellos el sistema no podría operar. Por ello esta tarea tendrá la prioridad más alta. Posteriormente, debemos procesar los datos para que estén listos para su envío. Ésta será la siguiente tarea en orden de prioridad. Finalmente las tareas con menor prioridad serán, la tarea de conexión WiFi, ya que no influye que la conexión tarde un cierto tiempo más en llevarse a cabo y el envío de los datos a través de WiFi. En la Tabla 4.1 se muestran las prioridades asignadas.

Tarea	Prioridad
Recepción de datos	3
WiFi	1
Procesado de datos	2
Envío de datos	1

Tabla 4.1: Prioridades de las tareas en Open ESP RTOS.

En los siguientes apartados se describe en detalle el funcionamiento de cada una de las tareas.

4.2.2 *Recepción de los datos*

Esta tarea es la encargada de obtener los datos del microcontrolador Arduino Nano mediante el bus SPI. El ESP 8266 inicia la comunicación enviando un carácter determinado. Cuando el Arduino Nano recibe la interrupción del bus SPI, comienza a colocar en su registro de datos SPI los valores de los datos de los sensores. El ESP 8266 realiza las transacciones SPI necesarias hasta que obtiene todos los datos, cerrando la última transacción enviando el valor nulo, para que el Arduino sea consciente del fin de la comunicación.

La trama de datos que se intercambian es la que se muestra en la Figura 4.3. Como se puede ver, esta cuenta con los ocho datos de los sensores láser, y con los 9 valores proporcionados por la IMU (giróscopo, acelerómetro y magnetómetro para los tres ejes).

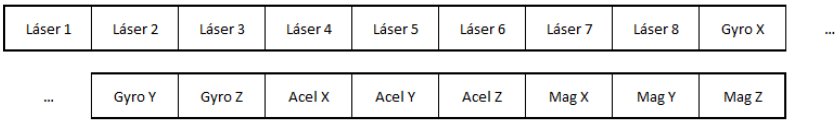


Figura 4.3: Estructura de la trama enviada en el bus SPI.

Estos datos requieren ser procesados, por lo que se reciben y se almacenan en la variable de datos compartida entre esta tarea y la tarea que lleva a cabo el procesado. Hay que destacar que esta variable esta protegida por semáforos para evitar problemas de condiciones de carrera.

Esta tarea es invocada con una frecuencia de 200 kHz, ya que los datos de giróscopo y acelerómetro se actualizan con esta frecuencia y son los que mayor tasa de actualización tienen. En la Tabla 4.2, se muestran los tiempos reales y frecuencia de ejecución medidos en la tarea.

Frecuencia (Hz)	t_{medio} (ms)	t_{max} (ms)
$200 \sim 200{,}04Hz$	$0{,}965ms$	$2{,}33ms$

Tabla 4.2: Tiempos medidos en la tarea de recepción de datos por el bus SPI.

4.2.3 *Conexión a red WiFi*

Esta tarea es la encargada de configurar el ESP 8266 para conectarse a la red WiFi deseada. Su funcionamiento es muy sencillo, simplemente trata de conectar a la red WiFi, durante varios intentos. Una vez consigue conectarse, esta tarea no tienen ninguna influencia en el sistema.

4.2.4 *Algoritmos de procesado de datos*

Como se ha comentado anteriormente, los datos recibidos por los sensores presentan ciertos inconvenientes, como puede ser ruido, valores erróneos, etc. Para solucionar estos problemas y tener unos datos con los que poder trabajar es necesario realizar un procesado de los mismos. En las siguientes secciones se explica el procesado llevado a cabo a los datos de los sensores, así como el impacto que este tiene en la adquisición de estos datos, si es el caso.

4.2.4.1 *Sensor láser de distancia VL53LoX*

Si observamos a los datos en crudo que el sensor VL53LoX obtiene midiendo la distancia con respecto a un objeto inmóvil (Figura 4.4),

obtenemos algo que difiere de lo esperado. A pesar de que la distancia que se está midiendo no varía, vemos que los datos recibidos si lo hacen, son ruidosos y pueden llevar al algoritmo a errores. Por esto es necesario filtrar estos datos, para trabajar con valores más constantes.

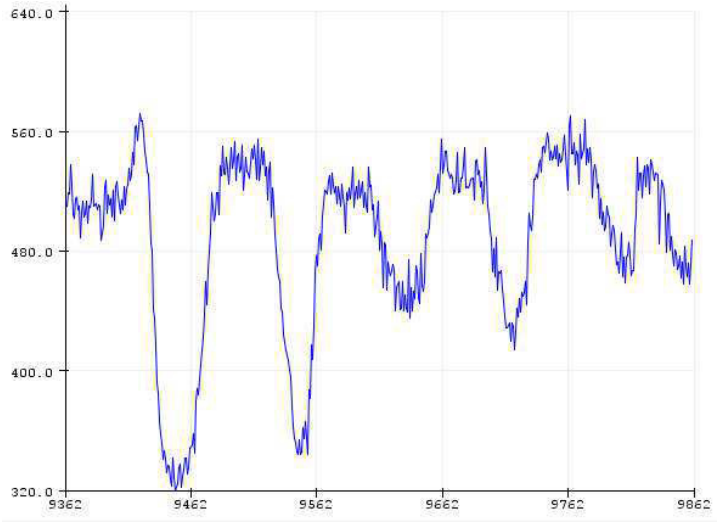


Figura 4.4: Datos proporcionados por el sensor VL53L0X para una distancia de 72 mm.

La solución a este problema es utilizar un filtro de Kalman. Se trata de un algoritmo desarrollado por Rudolf E. Kalman [8] que permite predecir el estado futuro de un sistema dinámico y contaminado por ruido blanco [9] [10]. Se trata de una de las mayores aportaciones al desarrollo tecnológico, ya que sin él hubiera sido prácticamente imposible la carrera espacial, entre otras muchas áreas en las que ha contribuido, como en navegación, en economía, en robótica, etc.

El modelo que utiliza este filtro es 4.1.

$$x_k = F_k x_{k-1} + B_k u_k + w_k \tag{4.1}$$

Donde F_k es la transformación del estado anterior al actual, x_k es el estado anterior, B_k es el modelo de control aplicado a la señal de control, u_k, y w_k es el ruido del proceso, que se asume normal de media cero.

El sistema se basa en un proceso recursivo que cuenta con dos pasos, predicción y corrección. A continuación se da una ligera idea de como funciona el filtro.

- **Predicción:**

  Consiste en obtener una estimación del estado en el que se encuentra el sistema a priori, sin tener en cuenta la observación

del estado actual, únicamente con los datos disposibles del estado anterior. En las ecuaciones 4.2 se indica su funcionamiento.

$$\begin{aligned} \hat{x}_{k|k-1} &= F_k \hat{x}_{k-1|k-1} + B_k u_k \\ P_{k|k-1} &= F_k P_{k-1|k-1} F_k^T + Q_k \end{aligned} \tag{4.2}$$

La predicción a priori del estado actual, $\hat{x}_{k|k-1}$, se obtiene transformando el estado anterior, $\hat{x}_{k-1|k-1}$, al estado actual mediante F_k y añadiendo el modelo de control, $B_k u_k$, en caso de que exista. La predicción a priori de la covarianza del ruido del proceso, $P_{k|k-1}$, se obtiene transformando la predicción de la covarianza del ruido del estado anterior, $P_{k-1|k-1}$, al estado actual mediante F_k y añadiendo la covarianza del proceso Q_k.

- **Corrección:**

  Una vez que se han recibido las observaciones del estado, es necesario corregir la estimación realizada a priori. En la ecuación 4.3 que se lleva a cabo es comprobar el error que se ha cometido en la estimación con respecto a la muestra recibida.

$$\begin{aligned} \hat{y}_k &= z_k - H_k \hat{x}_{k|k-1} \\ S_k &= R_k + H_k P_{k|k-1} H_k^T \end{aligned} \tag{4.3}$$

Donde $\hat{y}_k$ es el error cometido para el estado, z_k es la observación del estado actual, H_k es la transformación del estado al espacio de observaciones y R_k es la covarianza del ruido de la observacion.

A continuación, se va a corregir la estimación a priori a partir de la diferencia obtenida con respecto a la observación, pero, ¿Qué peso se le debe dar a esta diferencia? Esto se resuelve a traves de la ganancia de Kalman, que se puede ver en la ecuación 4.4.

$$K_k = P_{k|k-1} H_k^T S_k^{-1} \tag{4.4}$$

Esta ganancia es obtenida a partir de las matrices de covarianza del ruido del proceso y de la observación, ya que las matrices de covarianza representan cuánto varía esa magnitud y por ello son útiles para conocer el peso necesario para corregir la estimación correctamente. Una vez obtenida la ganancia Kalman, se aplica a la estimación a priori, en la ecuación 4.5.

$$\begin{aligned} \hat{x}_{k|k} &= \hat{x}_{k|k-1} + K_k \hat{y}_k \\ P_{k|k} &= (I - K_k H_k) P_{k|k-1} \end{aligned} \tag{4.5}$$

El filtro de Kalman ayudará en nuestra aplicación a proporcionar al algoritmo SLAM unos valores de distancia más estables, acordes con la realidad. Con el fin de obtener una buena estimación de los datos del sensor, es necesario llevar a cabo una medida de la varianza de las medidas que nuestro sensor lleva a cabo, esto es, la varianza de las observaciones. Se configura el filtro con las medidas comentadas anteriormente (la varianza de las medidas proporcionadas por nuestro sensor está en torno a 14) y en la Figura 4.5 se puede ver como los datos filtrados mejoran las medidas proporcionadas por el sensor.

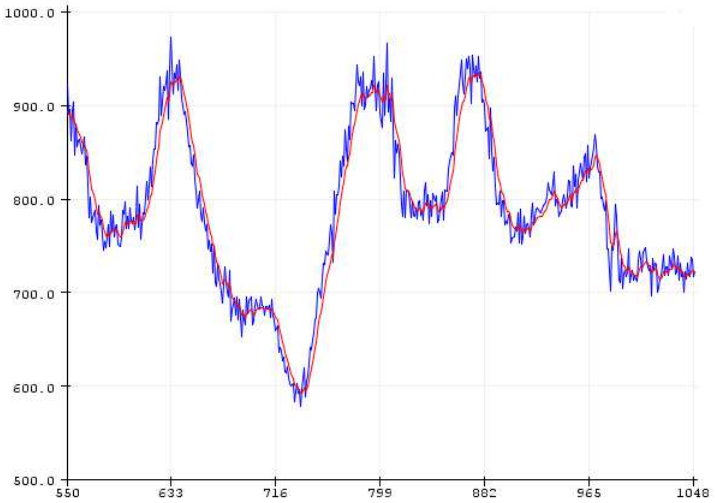


Figura 4.5: En azul los datos proporcionados por el sensor VL53LoX sin filtrar, en rojo, filtrados con Kalman.

La tasa de actualización de esta tarea debe ser la misma con la que le llegan los datos de los sensores láser. Esta es 33 Hz, en la Tabla 4.3, se muestran los tiempos y frecuencia de ejecución reales medidos en la tarea.

Frecuencia (Hz)	t_{medio} (ms)	t_{max} (ms)
$33{,}333 \sim 33{,}334Hz$	$0{,}175ms$	$0{,}67ms$

Tabla 4.3: Tiempos medidos en la tarea de aplicación del algoritmo de Kalman.

4.2.4.2 *Sensor MARG GY-87*

Los datos proporcionados por el sensor MARG, aceleración, velocidad de giro y valor del campo magnético, por sí solos no tienen utilidad para el dron. Es necesario realizar un procesado para poder extraer de ellos datos útiles.

La técnica que se requiere aplicar se conoce como algoritmo de fusión de datos. El objetivo es, a partir de los datos proporcionados por el sensor MARG, obtener la orientación del dron con respecto al sistema de referencia fijo de la tierra. Esta orientación es calculada en cuaterniones, un tipo de números similar a los números complejos, pero con dos unidades imaginarias más. Posteriormente, se transforman estos cuaterniones en ángulos de navegación, *roll*, Φ, *pitch*, θ, y *yaw*, Ψ, como se ve en la Figura 4.6.

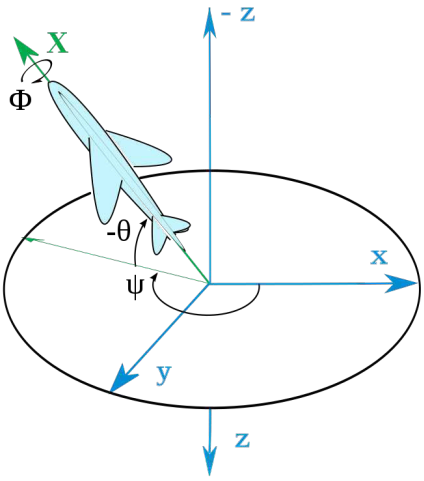


Figura 4.6: Angulos de navegación, *roll* , Φ, *pitch*, θ, y *yaw*, Ψ.

El giróscopo proporciona la velocidad de giro en los tres ejes, por lo que integrando esa velocidad de giro podríamos obtener la orientación del objeto. Sin embargo, esta integración genera errores (debido a las pequeñas vibraciones del giróscopo) que hacen que la orientación calculada tenga un deriva a medio y/o largo plazo. Aquí es donde comienzan a entrar en juego los algoritmos de fusión de datos. Estos errores pueden compensarse mediante el acelerómetro, que proporciona información del ángulo con respecto a la vertical que presenta el sensor. No obstante, el acelerómetro tiene el problema de que las medidas son muy ruidosas, debido a pequeñas vibraciones. Por lo que combinando ambas medidas, quedándonos con lo mejor de cada una, la componente en alta frecuencia para el giróscopo, y la componente en baja frecuencia para el acelerómetro, obtendríamos una medida más precisa. Esto se conoce como el filtro complementario [11], y es uno de los algoritmos de fusión de datos más simples y que menos coste computacional requieren.

Sin embargo, este filtro no es adecuado para nuestra aplicación, ya que solo utilizamos el acelerómetro y el giróscopo (seis grados de libertad), de los cuales, las medidas del acelerómetro no proporcionan información de *yaw* (rotación del eje z), por lo que, al tener únicamente una medida relativa al *yaw*, éste no puede ser corregido

y mantiene la deriva característica de las medidas obtenidas del giróscopo. Se requiere utilizar un sensor que proporcione una medida adicional, ortogonal a este eje, para de esta forma tener dos y que pueda ser corregida. Por ello se hace uso del magnetómetro. Este sensor proporciona información relativa a un sistema de referencia fijo (el campo magnético terrestre), por lo que ahora podremos corregir los errores del ángulo *yaw* con las medidas del magnetómetro, y obtener así una orientación relativa a un sistema de referencia fijo, la Tierra.

La mayoría de los métodos siguen esta idea anteriormente explicada. Existen multitud de algoritmos para llevar a cabo este cometido (diferentes en como calculan el error y/o lo minimizan), como por ejemplo, el filtro de Kalman, MultiWii, KK, Mahony, etc.

Para este proyecto se ha seleccionado el algoritmo de Madgwick [12], desarrollado en 2010 por Sebastian Madgwick. Se trata de un algoritmo de bajo coste computacional, código abierto, aplicable a IMUs de seis grados de libertad o a sensores MARG (nueve grados de libertad), ofrece unos buenos resultados y está optimizado para operar en pequeños microcontroladores, lo que lo hace ideal para nuestra aplicación.

El funcionamiento del algoritmo [13], se presenta en la Figura 4.7. En el grupo uno, vemos la corrección que se realiza a las medidas del magnetómetro para obtener el valor del campo magnético referenciado al sistema de referencia de la Tierra. De esta forma, el valor del campo magnético solo tiene componentes en el eje X e Y. Con estos datos y los proporcionados por el acelerómetro, se obtienen las correcciones mediante el algoritmo por descenso de gradiente. En el grupo dos, se lleva a cabo la corrección de la deriva del giróscopo. La ganancia β se encarga de minimizar los efectos de media cero, como ruido del sensor, errores de cuantificación, errores de calibración, etc. Una vez se tiene la estimación de la velocidad de giro corregida, se integra y normaliza, quedando como resultado los cuaterniones, que representan la orientación del sistema.

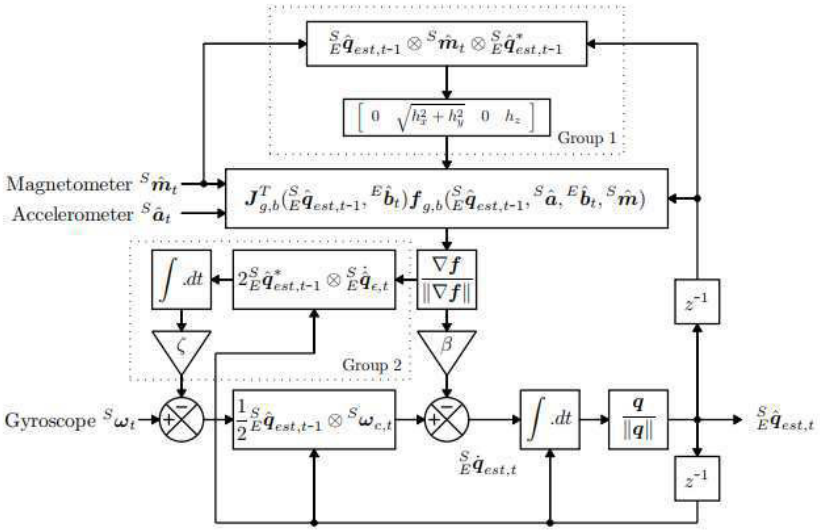


Figura 4.7: Esquema del algoritmo de Madgwick para un sensor MARG.

Como se ha comentado, este algoritmo se puede implementar en un microcontrolador de manera sencilla siguiendo los siguientes pasos:

1. Medir velocidad de giro , ω, aceleración , a, y campo magnético, m.
2. Calcular velocidad de giro únicamente con los datos del giróscopo y la orientación estimada anterior.
3. Tomar como referencia la dirección del campo magnético terrestre.
4. Aplicar el algoritmo por descenso de gradiente, teniendo en cuenta los datos de acelerómetro y giróscopo, para obtener las correcciones necesarias a la velocidad de giro.
5. Aplicar las correcciones a la velocidad de giro estimada inicialmente.
6. Integrar la velocidad de giro corregida.
7. Normalizar los cuaterniones.
8. Volver al paso 1.

Una vez se tienen los cuaterniones, lo único que hay que hacer es obtener los ángulos de navegación, *roll* , Φ, *pitch*, θ, y *yaw*, Ψ, a partir de estos con una sencilla operación. En este trabajo únicamente se necesita el valor *yaw*, Ψ, por lo que solo se obtiene este, con la ecuación 4.6.

$$\Psi = atan^2 \frac{q_w q_z + q_x q_y}{0{,}5 - q_y^2 - q_z^2} \tag{4.6}$$

Con el fin de trabajar con un error pequeño, es necesario iterar el algoritmo varias veces (en torno a 3 ó 4) con los mismos datos de entrada. Recordamos que la tasa de actualización de datos más rápida es 200 Hz, por lo que se ha decidido que la tasa de actualización del algoritmo sea de 1 kHz. En la Tabla 4.4, se muestran los tiempos y frecuencia de ejecución reales medidos en la tarea.

Frecuencia (Hz)	t_{medio} (ms)	t_{max} (ms)
$200 \sim 200{,}04Hz$	$0{,}520ms$	$1{,}458ms$

Tabla 4.4: Tiempos medidos en la tarea de aplicación del algoritmo de fusión de datos.

4.2.5 *Envío de datos*

Una vez que se cuenta con los datos correctamente procesados y listos para su uso en el algoritmo, es el momento de enviarlos al ordenador personal que llevará a cabo el algoritmo SLAM.

Esta tarea es la encargada de generar el paquete UDP y de transmitirlo a través de la red WiFi a la que estamos conectados. Primero se crea el *socket* mediante el cual se van a enviar los paquetes, asociado a la dirección *broadcast* de la red. Esto evita tener que configurar continuamente la IP si esta cambia en el ordenador que recibirá los datos, simplemente con estar en la misma red, obtendrá los paquetes si se recibe en el puerto adecuado. Tras esto, cada vez que se detecte que hay datos que enviar, la tarea coge estos datos, los formatea a texto (en la Figura 4.8 se muestra la trama), y los envía en un paquete UPD a través del socket. De nuevo, la variable que contiene los datos está protegida para que dos tareas no lean o modifiquen estos datos a la vez.

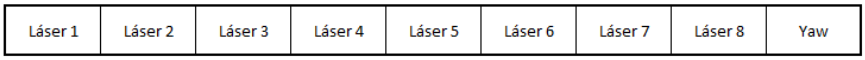


Figura 4.8: Estructura de los datos de la trama enviada a través de WiFi.

Esta tarea se lleva a cabo cada vez que se tiene un conjunto de datos nuevo. A pesar de que los datos tienen distinta tasa de actualización, es cada $30ms$ cuando se tiene un conjunto de datos nuevo completo, es decir, todos los datos estan actualizados, por lo que esta tarea tendrá una frecuencia de 33 Hz. En la Tabla 4.5, se muestran los tiempos y frecuencia de ejecución reales medidos en la tarea.

Tras esto, el ordenador personal encargado de llevar a cabo el algoritmo SLAM recibirá los datos y comenzará a procesarlos. En el

Frecuencia (Hz)	t_{medio} (ms)	t_{max} (ms)
$33{,}333 \sim 33{,}334 Hz$	$1{,}403 ms$	$5{,}277 ms$

Tabla 4.5: Tiempos medidos en la tarea de envío de datos por WiFi.

siguiente capítulo, se detalla en profundidad el algoritmo y su funcionamiento.

5

ALGORITMO SLAM

En este capítulo se explica qué es un algoritmo SLAM, su base teórica y varios de los métodos disponibles para llevarlo a cabo. Posteriormente, se detalla en profundidad el algoritmo SLAM utilizado, las diferentes funciones que debe implementar así como los datos de salida que proporciona el algoritmo para evaluar su funcionamiento.

5.1 FUNDAMENTO TEÓRICO

Como se ha comentado anteriormente, el objetivo del algoritmo SLAM es proporcionar un mapa del entorno donde está trabajando el sistema y, a partir de este mapa, estimar la localización de éste [14] [15]. Esto se consigue mediante los diferentes sensores, que analizan el entorno y ciertos parámetros necesarios del propio sistema.

Teóricamente, el problema del SLAM está solucionado. Mediante una aproximación probabilística es posible obtener un mapa del entorno conocido y, a su vez, posicionar al sistema en éste. Cuando nos movemos al plano práctico, se observa que aún quedan cuestiones abiertas, tales como obtener mapas fieles en entornos dinámicos, tener un algoritmo general que proporcione buenos resultados en escenarios muy diferentes, realizar SLAM en superficies muy grandes, entornos desestructurados, etc.

5.1.1 *Formulación matemática*

La formulación del problema se presenta en la ecuación 5.1

$$P(\mathbf{x}_k, \mathbf{m} | \mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0) \tag{5.1}$$

El estado actual, $\mathbf{x}_k$ y el mapa, $\mathbf{m}$, se estima a partir de las observaciones desde el instante inicial hasta el actual, $\mathbf{Z}_{0:k}$, de los controles (o medidas) de movimiento desde el estado inicial al actual, $\mathbf{U}_{0:k}$ y del estado inicial, $\mathbf{x}_0$.

Sin embargo, vemos que para obtener una estimación de la posición y del mapa, es necesario conocer todas las observaciones anteriores, $\mathbf{Z}_{0:k}$, y todos los controles anteriores, $\mathbf{U}_{0:k}$, y esto exige muchos recursos computacionales. Estimar el mapa y la posición a lo largo de toda la trayectoria se conoce como *full* SLAM. Sin embargo, el problema clasificado como *Online* SLAM, que es el que más popularidad tiene,

estima la posición y el mapa del estado actual, únicamente a partir del estado y observación anterior. Para llevar a cabo esto se realiza una marginalización de una variable. Como este problema anterior es similar a los filtros, a algunas soluciones del SLAM se les conoce como filtro.

A través del Teorema de Bayes, podemos conseguir, usando el estado anterior (en ecuación 5.2), un modelo de observación (en ecuación 5.3) y un modelo de movimiento (en ecuación 5.4), que el problema se convierta en recursivo.

$$P(\mathbf{x}_{k-1}, \mathbf{m} | \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k-1}) \tag{5.2}$$

$$P(\mathbf{z}_k | \mathbf{x}_k, \mathbf{m}) \tag{5.3}$$

$$P(\mathbf{x}_k | \mathbf{x}_{k-1}, \mathbf{u}_k) \tag{5.4}$$

Con esta nueva formulación, es posible estimar la posición y el mapa con las observaciones anteriores y el control del momento actual, como se ve en la ecuación 5.5.

$$\begin{aligned} P(\mathbf{x}_k, \mathbf{m} | \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k}, \mathbf{x}_0) = \int P(\mathbf{x}_k | \mathbf{x}_{k-1}, \mathbf{u}_k) \\ \times P(\mathbf{x}_{k-1}, \mathbf{m} | \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k-1}, \mathbf{x}_0) d\mathbf{x}_{k-1} \end{aligned} \tag{5.5}$$

Una vez se tiene esta estimación a partir del control proporcionado en el instante k, se corrige a través de las observaciones obtenidas en el mismo instante, k, como se muestra en la ecuación 5.6.

$$P(\mathbf{x}_k, \mathbf{m} | \mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0) = \frac{P(\mathbf{z}_k | \mathbf{x}_k, \mathbf{m}) P(\mathbf{x}_k, \mathbf{m} | \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k}, \mathbf{x}_0)}{P(\mathbf{z}_k | \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k})} \tag{5.6}$$

Con esto el problema queda resuelto parcialmente, ya que es necesario definir cómo es la función densidad de probabilidad del modelo de observación y de movimiento elegido. Aquí es donde entran en juego diferentes métodos para su resolución.

5.2 MÉTODOS SLAM

Existen multitud de aproximaciones para abordar este problema, a continuación se comentan algunos de ellos.

La aproximación más usada en la década de los 90 fue la de usar un filtro de Kalman extendido (EKF). Este método asume errores Gaussianos y utiliza modelos lineales para representar características no

lineales. Esto a veces puede llevar a presentar soluciones inconsistentes, si no se esta aplicando en un caso lineal, ya que solo es posible representar funciones densidad de probabilidad gaussianas, lo que limita la aplicación del sistema a cualquier otra función densidad de probabilidad.

Un método que introdujo un cambio de enfoque a este problema, con respecto al anterior, fue el filtro Rao - Blackwellised [16]. Este método se basa en el uso combinado de filtro de partículas y filtrado Kalman extendido. Representa modelos no lineales y no Gaussianos de distribución de la posición, lo que permite representar cualquier función densidad de probabilidad. Este método permite reducir el espacio de estados para el que se generan las partículas mediante la aplicación de Rao-Blackwellisation, y de esta forma, que sea viable computacionalmente.

DP-SLAM es un método aplicado a datos láser. Consiste en un filtro de partículas, en el que cada partícula tiene un mapa asociado. Este método se centra en cómo compartir partes del mapa por parte de varias partículas para hacerlo más eficiente.

Un método no basado en filtros es *Graph* SLAM. Este método resuelve el problema del *Full* SLAM ya que utiliza toda la formación desde el estado inicial. Se genera un grafo en el que los nodos son las posiciones y las marcas obtenidas, conectados mediante las observaciones, las que se toman como restricciones con un coste asociado. El problema consiste en optimizar la posición del robot y las marcas, en función del coste total. Es bastante costoso computacionalmente.

Existen multitud más como L-SLAM, Occupancy Grid SLAM, LSD-SLAM, MonoSLAM, etc.

5.3 ELECCIÓN DEL ALGORITMO SLAM

Como se comentó en la introducción, se va a seleccionar un algoritmo adecuado y a realizar ligeras modificaciones del mismo para poder utilizarlo en nuestro sistema.

A la hora de elegir el algoritmo es muy importante el tipo de datos de entrada, es decir, los sensores, con los que el algoritmo va a trabajar. Existen multitud de sensores/tipo de información que puede ser utilizada como entrada para el algoritmo SLAM. Por ejemplo, es común utilizar odometría, integrando la información del movimiento realizado por las ruedas, es posible obtener el movimiento que ha seguido el robot y, por tanto, determinar su posición, también es posible utilizar cámaras, que proporcionan imágenes de las que extraer puntos de referencia, sensores sonar, que proporcionan la distancia a

los objetos en las inmediaciones del sistema, sensores láser, similares a los sonar, y muchos más.

En nuestro sistema contamos con los datos proporcionados por los sensores láser, que nos indican la distancia a los objetos en los alrededores, y con los datos proporcionados por un sensor MARG, que nos permiten conocer la orientación del sistema y detectar rotaciones del mismo.

Teniendo esto en cuenta, se ha seleccionado un algoritmo SLAM libre, de código abierto que utiliza datos láser de distancia. Se trata de TinySLAM [17], un algoritmo desarrollado por Bruno Steux y Oussama El Hamzaoui, en el Centro de Robotica de París. El código puede encontararse en *openslam-org.github.io*.

Es un algoritmo sencillo escrito en C, de menos de 200 líneas, que utiliza datos de un escáner láser e información de odometría. El algoritmo almacena las observaciones en un mapa de rejillas de ocupación. Sin embargo, no almacena valores discretos, como existe obstáculo o no existe, por el contrario, almacena valores numéricos de forma que expresan la probabilidad de detectar un obstáculo en esa posición. Sumado a esto, el algoritmo no posiciona únicamente el obstáculo en la rejilla correspondiente a la posición donde lo detecta, sino que genera una especie de "agujero" centrado en esa rejilla, pero que deforma las rejillas de las inmediaciones, aumentando su probabilidad de encontrar un obstáculo en ellas. En la Figura 5.1 se puede ver una representación de cómo se deforman las rejillas cercanas a un obstáculo (en la posición $x = 0, y = 0$). La extensión de esta deformación viene determinada por un parámetro del algoritmo, como se verá más adelante.

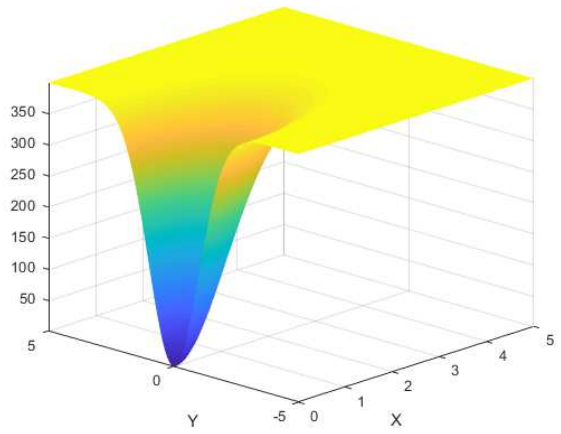


Figura 5.1: Representación de una zona del mapa generado por el algoritmo cuando encuentra un obstáculo $(x = 0, y = 0)$.

Para llevar a cabo la localización, utiliza un filtro de partículas. Se obtienen una cantidad de partículas (que representan una distribución de probabilidad) con diferentes posiciones del sistema. De todas las partículas, se obtiene la que mayor verosimilitud guarda con el mapa, y la posición que tiene esa partícula se guarda como posición del sistema.

Con el fin de adaptar el algoritmo a nuestro sistema, es necesario realizar varias modificaciones, ya que está pensado para operar en robots terrestres y nuestro objetivo es una aeronave ligera. En la siguiente sección se explica en profundidad el algoritmo, su estructura y como afronta y resuelve el problema del SLAM.

5.4 ESTRUCTURA DEL ALGORITMO

El esquema general que presenta el algoritmo usado es el que se ve en la Figura 5.2.

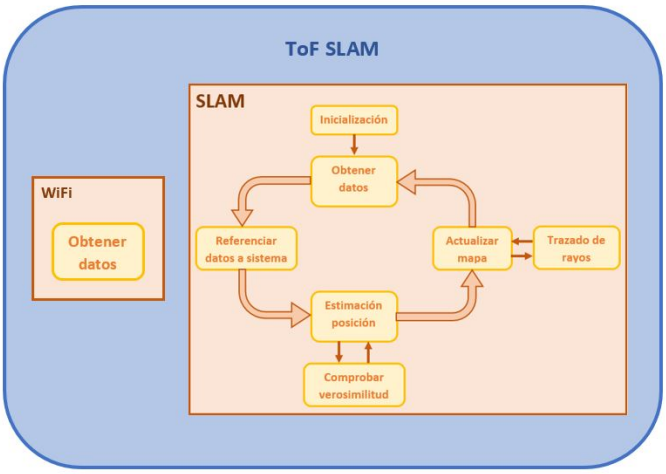


Figura 5.2: Esquema de bloques del algoritmo implementado.

Como se puede ver, existen dos procesos, el proceso denominado WiFi, que es el encargado de obtener los datos que el sistema de adquisición envía a través de la red WiFi. Este proceso obtiene los ocho datos proporcionados por los sensores láser, así como la orientación, yaw, ψ, del sistema y los almacenará en la estructura de datos creada.

El otro proceso es el algoritmo SLAM. Los datos entre los procesos se comparten a través de la estructura, protegida mediante un semáforo. El algoritmo SLAM iterará cada vez que reciba nuevos datos. Se trata de un algoritmo muy sencillo y de bajo coste computacional, por lo que este podrá ejecutar cada iteración en el tiempo entre set de datos (33,3ms), como se detallará más adelante.

El bloque de inicialización únicamente crea las variables necesarias y asigna los valores predeterminados. Tras esto, se espera por nuevos datos, que una vez que llegan, es necesario posicionarlos con respecto al sistema. Tras esto, se realiza una estimación de la posición en la que se encuentra el mismo, utilizando un mapa con las observaciones anteriores y un filtro de partículas. Una vez se tiene la nueva posición, se actualiza el mapa con la posición de las observaciones obtenidas en el instante actual y se espera por nuevos datos.

A continuación, se detallan en profundidad cada una de las funciones implicadas en el algoritmo.

5.4.1 *Inicialización*

El primer paso necesario es llevar a cabo la inicialización del algoritmo. Esto se lleva a cabo en la función *ts_map_init*, encargada de crear las variables necesarias con sus valores iniciales y configurar los diferentes parámetros del algoritmo para que el sistema obtenga los resultados correctos.

- **Mapa:** Se requiere seleccionar el tamaño del mapa que se va a generar, así como iniciar todos sus valores y definir los valores máximo que se van a permitir en las funciones de distribución que el mapa representa (se comenta en profundidad más adelante).
- **Láser:** Es necesario configurar ciertos parámetros de los láser para que el sistema pueda interpretar los datos que éste le proporciona, como el número de láseres que vamos a utilizar, su posición con respecto al centro del sistema, el ángulo que forman con respecto a la orientación predeterminada y la máxima distancia que proporcionan.
- **Estado:** El estado almacena información referente a la posición del sistema y ciertos valores de configuración utilizados en los bloques de estimación de la posición, como la varianza de las partículas generadas en el filtro, el tamaño del "agujero", que es cuánto se expande la función densidad de probabilidad de un obstáculo detectado, y la memoria del algoritmo, que representa la importancia que el algoritmo asigna a las observaciones actuales, con respecto a las pasadas.

Una vez se ha configurado todos los valores, el algoritmo puede comenzar a operar.

5.4.2 *Obtención de datos*

Tras la inicialización del algoritmo, el siguiente paso requiere de obtener los datos para comenzar a operar. La función *ts_get_data* se encarga de esto. Para notificar que existen nuevos datos se utiliza un *flag*. Es decir, el algoritmo espera a que este *flag*, protegido por un semáforo, se active, indicando que existen nuevos datos para ser procesados, entonces accede a estos datos y los transfiere a una variable propia, liberando el *buffer* para que sea posible recibir nuevos datos.

Los datos recibidos son las ocho distancias que existen entre los láser y los objetos que tienen delante, y la información de orientación. La orientación del sistema, se utilizará como un valor determinista, ya que es obtenida a través de un algoritmo de fusión de datos que utiliza nueve medidas diferentes, y se considera lo suficientemente precisa para la aplicación.

5.4.3 *Referenciar datos a sistema*

Una vez se han obtenido los datos, es posible aplicar el algoritmo. No obstante, estos datos proporcionan información de la distancia entre el láser y el objeto, si es el caso, que tienen delante. Por lo que para poder realizar SLAM, hay que referenciarlos al propio sistema. Esto se lleva a cabo en la función *ts_build_scan*. En la Figura 5.3 se puede ver el sistema de referencia elegido para nuestra aplicación.

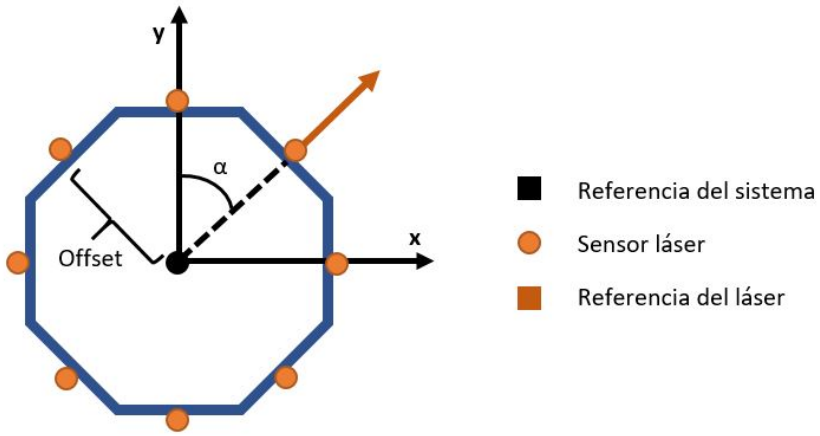


Figura 5.3: Diagrama que muestra los sensores láser y el sistema con sus ejes de referencia.

Conociendo la localización de cada uno de los sensores láser, esto se puede llevar a cabo de una manera muy sencilla a través de tri-

gonometría. Esta función implementa esta operación, descrita en las ecuaciones 5.7.

$$\begin{aligned} x &= (d + offset)cos(\alpha) \\ y &= (d + offset)sin(\alpha) \end{aligned} \tag{5.7}$$

Además, esta función también indica si estas medidas representan la detección de un objeto, o por el contrario, no existe ningún objeto en la dirección que apuntan y rango de medición.

5.4.4 *Estimación de la posición*

Cuando se han referenciado al sistema los últimos valores proporcionados por los sensores, podemos tratar de estimar la posición de éste buscando en el mapa que se tiene del entorno si estas detecciones se encuentran en él, y de esta forma, obtener la posición. La función que se ocupa de esta tarea es *ts_monte_carlo_search*.

El objetivo de esta función es obtener, mediante el método Monte Carlo, la posición que mas aproxime las observaciones del momento actual con las detecciones que existen el mapa que se tiene del entorno. Para ello utiliza un filtro de partículas. Cada nueva partícula contiene una posición x e y.

Estas partículas se generan aleatoriamente de acuerdo a una distribución normal y a una varianza que hemos establecido en la inicialización del algoritmo, utilizando únicamente números enteros, lo que lo hace muy sencillo en términos de carga computacional. El algoritmo generador de números aleatorios utilizado es SHR3 (registro de desplazamiento de 3 bits).

Para obtener una estimación de la posición que tiene el sistema en ese instante, es necesario analizar la verosimilitud de cada una de las partículas generadas (pueden ser vistas como posiciones) con el mapa que se tiene del entorno. El proceso que se sigue para obtenerlo, se describe a continuación.

5.4.4.1 *Comprobación de la verosimilitud*

El filtro de partículas, genera una función de distribución. Con la comprobación de la verosimilitud, lo que hacemos es quedarnos con aquella que tiene mayor probabilidad. Se lleva a cabo en la función *ts_distance_scan_to_map*.

Esta función compara la observación actual, tomando como posición la proporcionada por la partícula, con el mapa que se tiene. Para hacer esto parte de unas observaciones y la posición de la partícula

bajo *test*. Posiciona los datos de los sensores con respecto a la posición de la partícula, referenciados en las coordenadas del mapa global. Una vez que se tiene la posición absoluta (x_i, y_i) de las observaciones, suponiendo que el sistema está en la posición proporcionada por la partícula, se suman los valores del mapa en las coordenadas (x_i, y_i) de cada componente de la observación.

Como se ha comentado anteriormente, este mapa almacena en cada rejilla de ocupación la probabilidad de que ahí exista un obstáculo representando con un valor pequeño una alta probabilidad de obstáculo, y con un valor alto, una baja probabilidad. Por ello, podemos usar este método para comprobar la verosimilitud de la posición proporcionada por la partícula. Si la posición proporcionada por la partícula es muy próxima a la posición real del sistema, la suma de los valores que tiene el mapa en las posiciones donde suponemos que están los obstáculos (x_i, y_i), será un valor bajo. Por el contrario, si la posición de la partícula no se asemeja con la posición real y en las inmediaciones de esta no hay obstáculos, obtendremos un valor alto.

De esta forma, es posible obtener, de un conjunto de partículas suficientemente grande, cual es la que mejor representa la posición real del sistema, y por tanto, puede ser usada como estimación de la posición.

Cabe destacar, que en esta función, cuantas más observaciones contribuyan con obstáculos detectados, más verosimilitud se otorgará a la partícula. Esto se debe a que varios o todos los láser pueden no detectar ningún obstáculo y, por lo tanto, proporcionan menos información al sistema.

5.4.5 *Actualización del mapa*

Llegados a este punto del algoritmo, ya se cuenta con una posición estimada de donde se encuentra el sistema, no obstante, las observaciones de este instante no han sido aún añadidas al mapa. La función del algoritmo lleva a cabo esta tarea es *ts_map_update*.

En cuanto a la actualización del mapa, es necesario definir dos parámetros que se ven involucrados en este bloque, uno es el tamaño del agujero, es decir, cuánto se va a expandir en el mapa la deformación provocada por la detección de un obstáculo, y el otro es la memoria del mapa, es decir, cómo de importantes son las nuevas muestras con respecto a las que ya están en él. Este último parámetro va a definir el rendimiento del algoritmo, ya que una memoria baja hace que el sistema sea muy reactivo a cambios en el entorno, pero tendrá más incertidumbre ya que no recordará referencias obtenidas en el pasado, y un sistema con más memoria, trabajará con menos incertidumbre,

pero será difícil detectar eventos dinámicos y no estacionarios (como personas moviéndose alrededor).

El primer paso que realiza este bloque es obtener las coordenadas referenciadas al mapa global de cada una de las observaciones. Como se ha comentado anteriormente, las observaciones que contienen la detección de un objeto no se representan en un mapa únicamente en la coordenada en la que han sido detectados, sino que se realiza un "agujero" en esa coordenada, pero las rejillas de alrededor también se ven deformadas, es decir, se representa la detección como una función de probabilidad y no como algo determinista. Por ello aquí también se calculan las coordenadas de la rejilla hasta donde afecta esta deformación. Con todo esto, se llama a la función que se encarga de trazar los rayos entre la detección y el sistema.

5.4.5.1 *Trazado de rayos*

Una vez se tiene posición del sistema, posición del obstáculo, y posición hasta donde se ve modificado el mapa por el obstáculo, todo ello referenciado al sistema de referencia global (el del mapa), es necesario trasladar al mapa el estado de todas las rejillas de las que se tiene información. Además de las coordenadas calculadas mencionadas anteriormente, también se sabe que las rejillas bajo la línea que une el sensor, y la detección (o no detección), están libres de obstáculos, por lo que es necesario un método que recorra todas estas rejillas y les asigne el valor deseado. Para llevar esto a cabo se utiliza la función *ts_map_laser_ray*.

Esta función implementa un algoritmo de Bresenham [18]. Este algoritmo se encarga de una manera muy eficiente y rápida, sin utilizar enteros, de obtener una línea entre dos puntos en un mapa de dos coordenadas, como se muestra en la Figura 5.4.

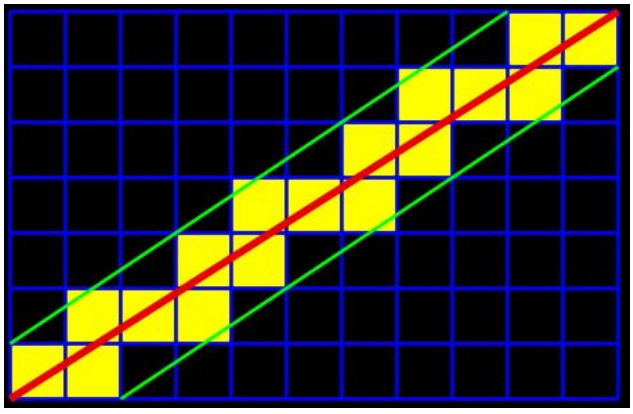


Figura 5.4: Demostración del algoritmo de Bresenham, donde la línea roja es la deseada y los cuadros amarillos el resultado.

En este bloque, también se calcula el valor de la función densidad de probabilidad que asigna a cada una de las rejillas involucradas, que se encuentran en el interior del "agujero" creado por el obstáculo detectado. Queda pendiente cómo introducir el valor de cada una de las rejillas al mapa teniendo en cuenta los valores almacenados previamente en ella (de observaciones anteriores). Para actualizar el estado de estas se sigue la ecuación 5.8.

$$x_k = (1 - \alpha)x_{k-1} + \alpha z_k \tag{5.8}$$

El valor actual de la rejilla, x_k, se compone de el valor anterior, x_{k-1}, y el valor asignado en la observación actual, z_k, ponderado por la memoria del algoritmo, α. Gracias a este último parámetro podemos definir cual es la reactividad del algoritmo.

Con esto el algoritmo integra los valores de la observación en el mapa, y se puede obtener la siguiente observación para repetir todo el proceso de nuevo.

5.5 DATOS DE SALIDA

Con el fin de evaluar el rendimiento del algoritmo, es necesario obtener algún dato que nos proporcione información de cómo está funcionando éste. Para ello, el algoritmo proporciona los siguientes datos con las funciones *ts_save_map_pgm* y *ts_save_position*.

- **Mapa:** Se trata de una representación del mapa obtenido, donde el color blanco representa una probabilidad nula de encontrar un obstáculo y el color negro una probabilidad del 100 % de encontrar un obstáculo.
- **Posición actual:** Estimación de la posición en la que se encuentra el sistema.
- **Trayectoria:** Posiciones consecutivas seguidas por el sistema desde el inicio de la adquisición.

Para representar estos datos, se ha realizado un *script* en *Python*. Con la ayuda de la biblioteca `pygame`, una biblioteca para desarrollo de videojuegos 2D, representamos el mapa, la posición actual del sistema y la trayectoria. El *script* creado se llama *map.py*. El *script* nos muestra el mapa que se está generando y permite activar y desactivar en tiempo real mediante el teclado la posición, la trayectoria seguida por el sistema y las partículas que genera el filtro. En la Figura 5.5 se puede ver cómo es la pantalla generada por el *script*, donde en escala de grises vemos el mapa, la flecha roja incida donde está y qué orientación tiene el sistema, y en azul vemos la trayectoria seguida por este.

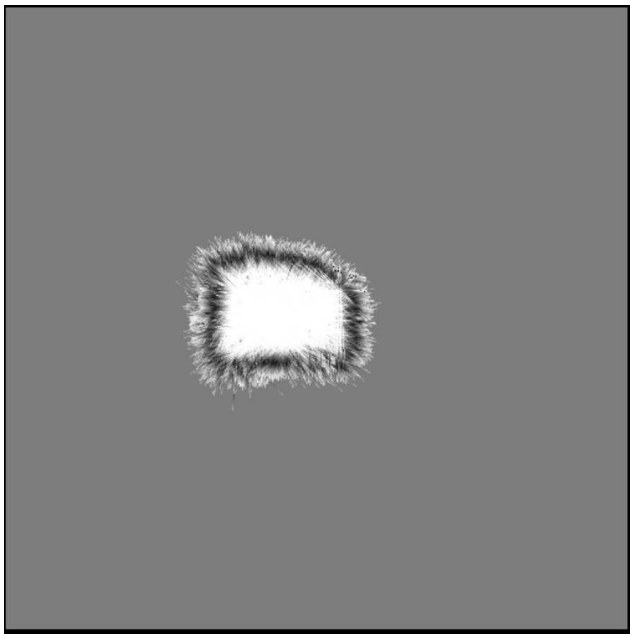


Figura 5.5: Captura del mapa generada por el *script* de generación de resultados.

Estos resultados son utilizados en el próximo capítulo para presentar el rendimiento del algoritmo.

6 RESULTADOS

En este capítulo se presentan los resultados obtenidos con el sistema implementado. Se comienza presentando los diferentes parámetros configurables del algoritmo y su impacto en los resultados obtenidos. Posteriormente se evaluará el algoritmo en cinco escenarios diferentes, aumentando el nivel de complejidad de estos.

Los resultados se presentan con imágenes que muestran el mapa creado, la posición del sistema y la trayectoria seguida por éste. Además, con el fin de poder mostrar el correcto funcionamiento del sistema, se adjuntan vídeos en los que se puede ver en tiempo real cómo se construye el mapa a medida que se mueve el sistema por el entorno y cómo es capaz de localizarse correctamente.

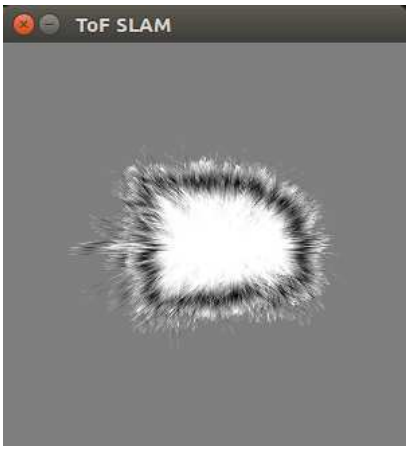
6.1 PARÁMETROS MODIFICABLES

En este apartado se va a mostrar cómo afectan cada uno de los parámetros modificables del algoritmo a los resultados obtenidos por el sistema. Se explica qué representan y se documenta con imágenes su efecto en los resultados.

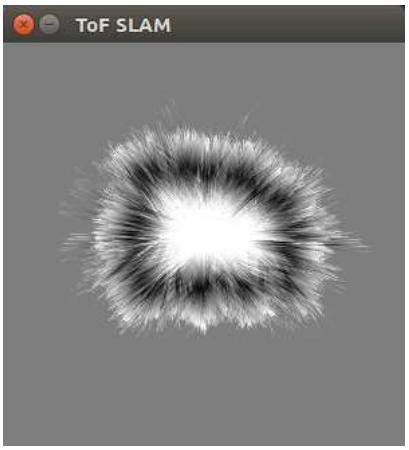
6.1.1 *Ancho de función de probabilidad de obstáculo*

Como se ha comentado en el capítulo anterior, el algoritmo no detecta un obstáculo como un objeto puntual, sino que asigna a éste una función densidad de probabilidad, creando un área donde es posible encontrar el obstáculo. Con este parámetro es posible modificar el ancho de esa área donde es posible encontrar el objeto.

Continuando con la analogía que se hacía en el capítulo anterior, si vemos estas funciones densidades de probabilidad como ”agujeros”, el punto más profundo se corresponde con una mayor probabilidad de encontrar el obstáculo. Estas deformaciones actúan como guías para el algoritmo, ya que cuando coinciden con la observación proporcionan una mayor verosimilitud que los puntos donde el mapa es llano. Por lo que, cuanto mayor sea el ancho de estos ”agujeros” más probabilidad tendrá el algoritmo de converger. Sin embargo, ésto presenta un problema, y es que cuanto mayor son estos ”agujeros”, más difícil es distinguir entre dos obstáculos cercanos, debido a la mayor incertidumbre se su posición.



(a) Ancho de función de probabilidad de obstáculo = 25 *cm*.



(b) Ancho de función de probabilidad de obstáculo = 50 *cm*.

Figura 6.1: Mapa generado en un entorno de 120×95 *cm* con diferentes valores del parámetro ancho de función de probabilidad de obstáculos.

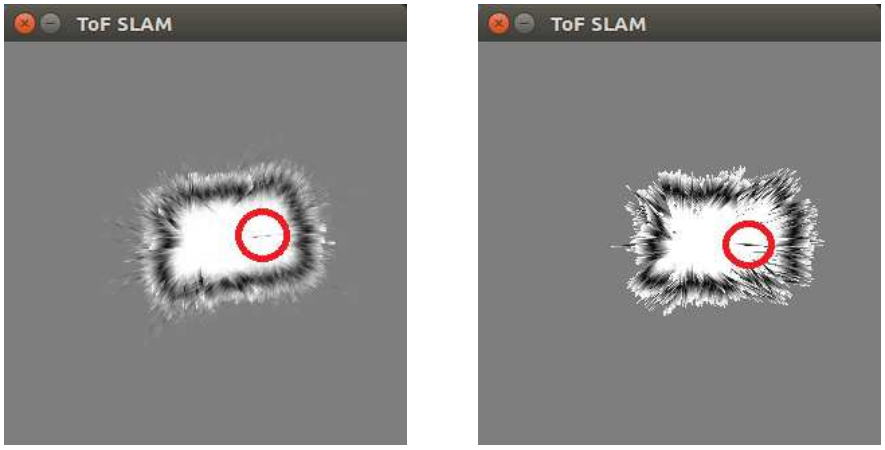
Debido a esto, es necesario encontrar un valor que ofrezca una solución de compromiso entre convergencia y resolución de obstáculos. Para esta situación no existe una solución única, ésta dependerá del tamaño de los obstáculos y del tamaño del entorno en el que se esté intentando obtener el problema. No obstante, durante todas las pruebas llevadas a cabo, se ha observado que el valor que se debe dar a este parámetro para que el sistema funcione correctamente debe ser al menos de 25 *cm*. Si se usa un valor inferior, el sistema no es capaz de converger y, por lo tanto, ni se obtiene el mapa ni la posición. En la Figura 6.1 se ve una comparativa de cómo funciona el sistema con diferentes valores de este parámetro en un mismo entorno. Éste parámetro se ajusta en cada uno de los escenarios.

6.1.2 *Parámetro de ajuste del algoritmo*

El algoritmo basa la estimación de la posición en el mapa que ha ido construyendo previamente. Una vez que tiene la posición estimada, actualiza el mapa con la información obtenida por los sensores en ese instante, pero, ¿cómo debe afectar esta nueva información al mapa que se ha construido con todas las observaciones anteriores? Esto lo define el parámetro de ajuste del algoritmo, conocido por la letra α. Este parámetro define cuánto impactarán las nuevas muestras en el mapa que se tiene. Si se tiene un parámetro bajo, las nuevas muestras modificarán el mapa, pero con un peso menor, sin embargo, si este parámetro es alto, las muestras obtenidas en la observación actual su peso en el mapa obtenido será mayor.

En términos de efecto sobre el resultado, este parámetro define la reactividad del sistema. Con un parámetro bajo el sistema necesita obtener una mayor cantidad de observaciones de un evento para que finalmente el mapa lo represente. Sin embargo, si el evento que se ha capturado es dinámico, es decir, desaparece a las pocas observaciones, el mapa no llegará a representarlo y no se tendrá en cuenta. Esto también aplica cuando se inicia el sistema, es decir, este parámetro también define la duración de la etapa de reconocimiento que debe hacer el sistema al iniciarse, ya que no tienen ningún tipo de información del entorno.

En la Figura 6.2 se muestran dos adquisiciones en las que existe un evento dinámico (se introduce un obstáculo) y se configuran dos valores diferentes de este parámetro. Se muestra el mapa obtenido después de insertar el obstáculo, y se observa como la adquisición con menor α, es decir, menor reactividad, muestra el obstáculo con una probabilidad mucho menor que el más reactivo. Incluso en las figuras se observa cómo el mapa de la adquisición con $\alpha = 80\,\%$ es más ruidoso, ya que el valor instantáneo tiene mucho más valor que los obtenidos en observaciones pasadas.



(a) $\alpha = 10\,\%$. (b) $\alpha = 80\,\%$.

Figura 6.2: Detección de un obstáculo dinámico con diferentes valores de α.

En en una aplicación real, es necesario que el sistema sea lo suficientemente reactivo como para incluir en el mapa obstáculos nuevos que han aparecido en el escenario, pero no demasiado, porque podría incluir en él eventos dinámicos del entorno que no sean característicos de éste, como por ejemplo, personas andando alrededor. Además, como se ve en la Figura 6.2, si α es grande, la estimación del mapa es más ruidosa. Es necesario remarcar que nuestro sistema tiene únicamente ocho medidas de láser en cada instante de tiempo, por lo que es necesario rotarlo para obtener un mapa en los 360 °, y la rotación de éste se realiza manualmente, por lo que no se puede hacer

con gran precisión. Si se usa un parámetro α muy bajo, se necesitarán muchas muestras para incluir los obstáculos en el mapa y ciertos ángulos no quedarían con suficiente información. Por esto, tras examinar el funcionamiento del sistema se ha seleccionado un valor de $\alpha = 20\,\%$. Este valor proporciona una buena reactividad del sistema y una correcta estimación del mapa generado.

6.1.3 *Varianza del filtro de partículas*

Como se ha comentado anteriormente, el método para obtener la estimación de la posición se lleva a cabo utilizando un filtro de partículas. Dicho filtro de partículas genera partículas con posiciones aleatorias, para que, de entre todas ellas, se elija la partícula que proporciona una posición con mayor verosimilitud con respecto al mapa que se tiene.

Existe un parámetro personalizable para este filtro, la varianza que otorga al proceso aleatorio de creación de partículas. Esto es, cuánto se van a expandir en el espacio, con respecto a la posición anterior, las partículas que se evaluarán como posibles posiciones actuales del sistema. En la Figura 6.3 se puede ver una comparativa entre una varianza muy pequeña, que prácticamente reparte las nuevas partículas muy cerca de la posición anterior, y una muy grande, que expande las partículas por casi todo el mapa.

(a) Varianza pequeña. (b) Varianza mayor.

Figura 6.3: Diferentes varianzas con las que se generan las partículas.

Esta parámetro va a tener un gran impacto en la obtención de la trayectoria que va a realizar el sistema. Si la varianza de las partículas es muy grande vamos a ver cómo el sistema va a realizar movimientos muy bruscos y tendrá mayor facilidad para perderse, pero si ésta es muy pequeña, va a ser difícil para el algoritmo seguir los movimientos del sistema.

Debido a la velocidad de movimiento de nuestro sistema, se ha seleccionado una varianza similar a la que se observa en la Figura 6.3 en la imagen (a).

6.2 PRUEBAS EN ESCENARIOS

En este apartado se muestran los resultados de las pruebas que se han llevado acabo en los diferentes escenarios que simulan situaciones reales en las que el sistema podría operar. Estas pruebas consisten en mover manualmente el sistema por el entorno simulado.

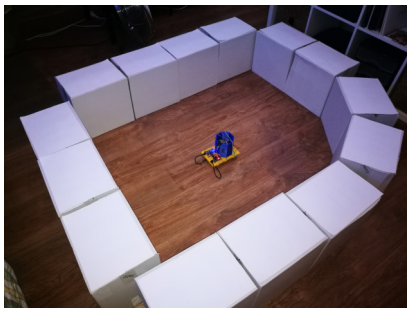
El algoritmo SLAM parte sin ningún conocimiento del entorno en el que está operando, por lo que nada más iniciar el sistema no es posible que obtenga instantáneamente la localización correcta. Por ello es necesario realizar una fase de reconocimiento del entorno. El sistema necesita acumular el mayor número posible de referencias para poder estimar una localización con respecto a éstas.

Tras varias pruebas con el sistema implementado, se ha llegado a la conclusión de que el mejor método para llevar a cabo esta fase son las rotaciones del sistema sobre su eje desde diferentes puntos. Es decir, comenzar rotando el sistema en la posición inicial, desplazarse en x o y una determinada distancia y volver a rotar, y así sucesivamente unas cuantas veces. Esto permite al sistema contar con suficientes referencias para que la localización converja. Una vez se ha completado el reconocimiento del entorno, el sistema proporciona una localización correcta dentro de éste.

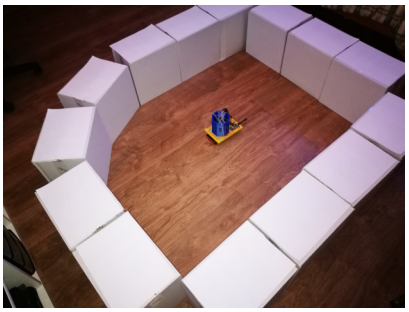
En los resultados, se presenta el escenario en cuestión. Con el fin de evaluar el algoritmo, se proporcionan imágenes del proceso de reconocimiento del entorno y, seguido a esto, se muestra cómo el sistema estima correctamente su localización en el entorno y se muestra la trayectoria seguida. Se adjuntan también *links* a vídeos, para que se pueda ver como se realiza el SLAM en tiempo real.

6.2.1 *Escenario 1: Sencillo*

El primer escenario que se va a analizar simularía una habitación sin obstáculos y sin eventos dinámicos. Se trata de un escenario sencillo, en el que el único objetivo es conseguir obtener el mapa y posicionar el sistema correctamente. En la Figura 6.4 se puede ver el escenario. las medidas del escenario son 125×100 cm.



(a)

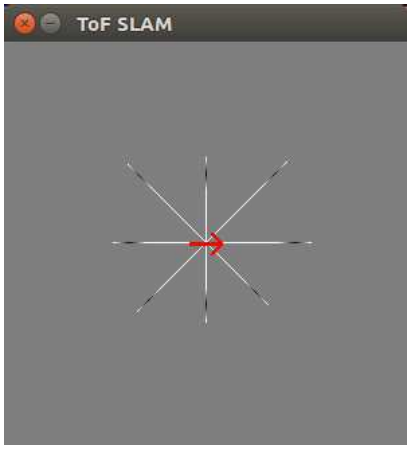


(b)

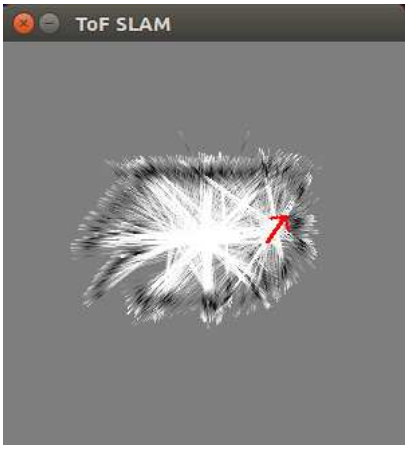
Figura 6.4: Escenario 1.

El valor σ tiene que configurarse a 25 cm ya que sino el sistema no es capaz de converger.

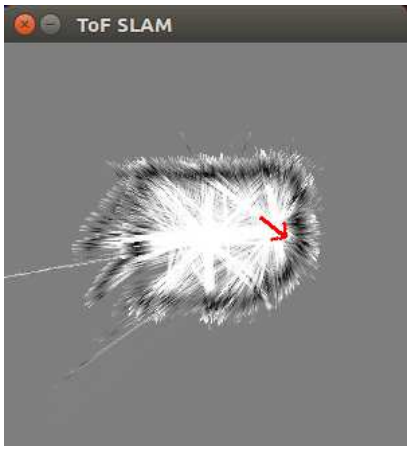
En la Figura 6.5 se ve el proceso de reconocimiento del escenario. Como se indica, en esta etapa de reconocimiento, primero se realiza una rotación sobre el sistema, posteriormente se adelanta éste, y se vuelve a rotar sobre sí mismo. Se mueve el sistema en el otro eje y se vuelve a rotar, para finalmente llevarlo a la zona por la que no había pasado y volverlo a rotar.



(a) Estado inicial.

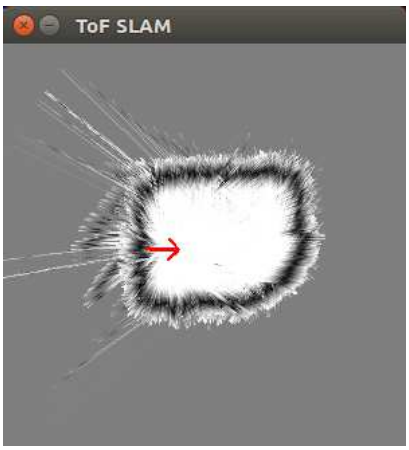


(b) Se rota el sistema sobre su eje.



(c) Movimiento lineal en un eje y rotación.

(d) Movimiento lineal en el otro eje y rotación.

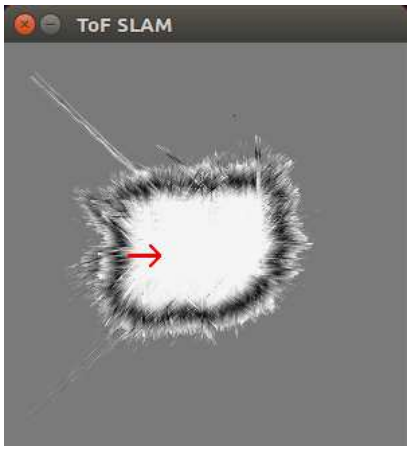


(e) Movimiento en ambos ejes y rotación.

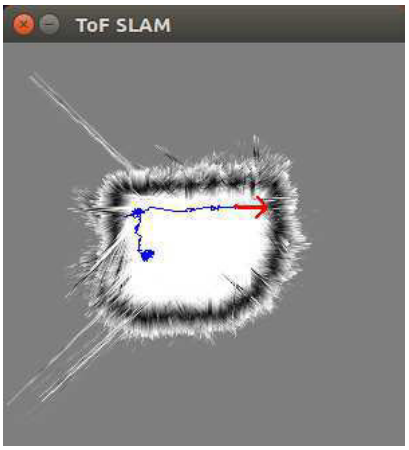
Figura 6.5: Reconocimiento del entorno en el escenario 1.

Como se ve, se obtiene el mapa del escenario que se está estudiando. Se puede observar que el mapa es bastante fiel al escenario. Ahora es el momento de analizar el funcionamiento del sistema en lo referente a la obtención de la posición.

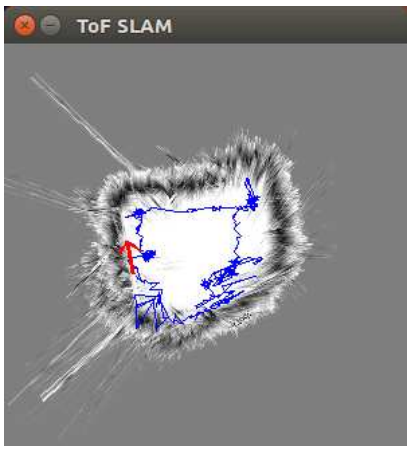
En la Figura 6.6 se muestran capturas del proceso de localización con el mapa obtenido anteriormente. Se muestra la posición inicial y los varios movimientos intermedios del sistema, para finalmente colocarlo en la posición central del mapa.



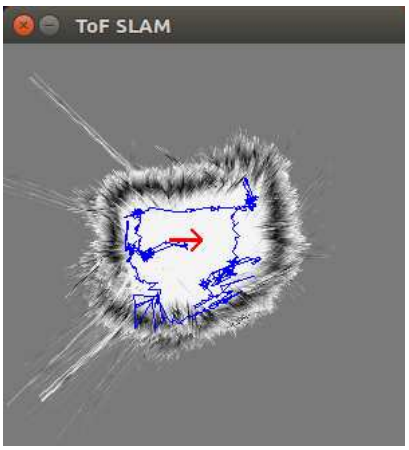
(a) Estado inicial.



(b) Movimiento hacia la izquierda y hacia adelante.



(c) Vuelta al punto inicial con rotaciones incluidas.



(d) Movimiento hasta el centro del mapa.

Figura 6.6: Localización del sistema en el escenario 1.

En el siguiente *link*, se puede ver un vídeo del sistema en funcionamiento. Como se puede ver con los resultados anteriores, el sistema funciona realmente bien en este escenario. Es capaz de obtener un mapa bastante fiel a la realidad, y posicionar el sistema dentro de éste, incluso responder a movimientos bruscos.

https://www.youtube.com/watch?v=cUtoO3gdK6k

6.2.2 *Escenario 2: Medio*

El siguiente escenario analizado es un entorno estático, algo más complejo que el primero, ya que uno de sus límites no es una superficie completamente lista, tiene una forma no rectangular y además cuenta con varios obstáculos distribuidos por él. El tamaño del mapa es 170×150 *cm*. En la Figura 6.7 se muestra el escenario.

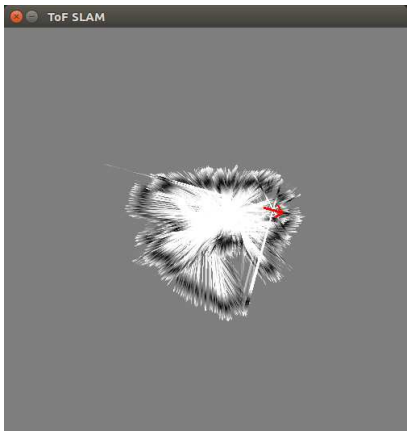
(a) (b)

Figura 6.7: Escenario 2.

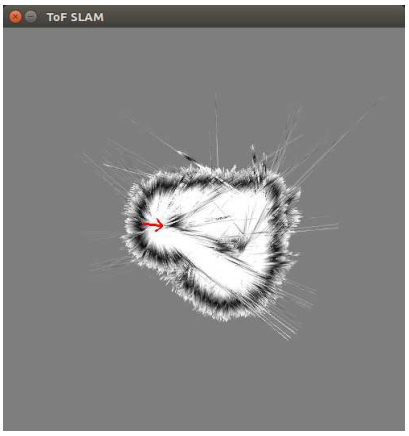
Como este mapa es más grande que el anterior, es necesario aumentar un poco el valor de σ, configurandolo a 30 cm. Esto va a tener un efecto adverso en los obstáculos, ya que son menores que el valor de σ asignado, pero es el valor necesario para que el algoritmo converja.

En la Figura 6.8 se ve el proceso de reconocimiento del escenario. Se mueve el sistema por el entorno, realizando rotaciones sobre sí mismo.

(a) Estado inicial. (b) Se rota el sistema sobre su eje.



(c) Movimiento lineal en un eje y rotación.

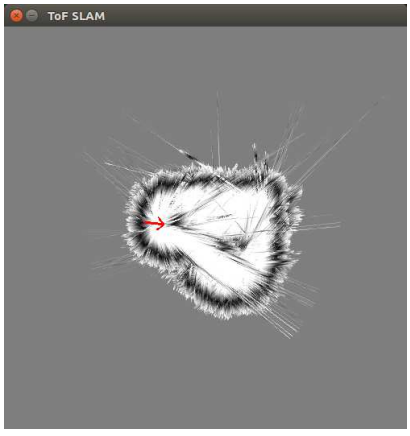


(d) Mapa obtenido.

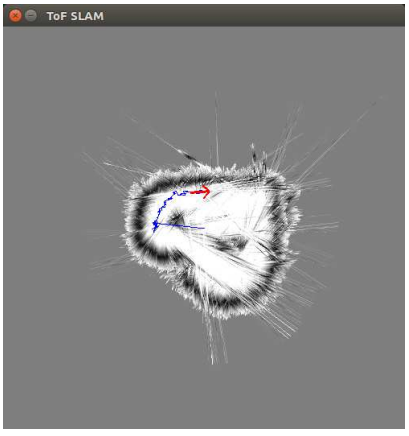
Figura 6.8: Reconocimiento del entorno en el escenario 2.

Como se puede ver, el mapa se reconstruye correctamente, vemos que los obstáculos también han sido obtenidos por el algoritmo en la posición correcta. Como se comentaba anteriormente, en los obstáculos se ve que la función densidad de probabilidad de extiende más del espacio que ocupa el propio obstáculo, debido al valor de σ.

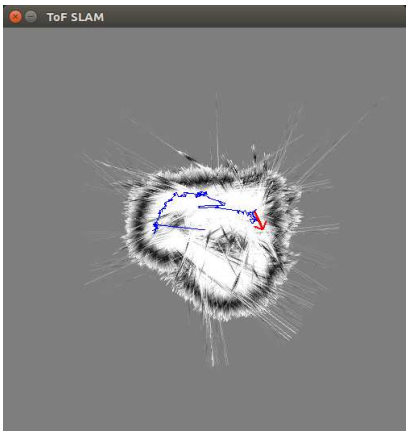
En cuanto al proceso de localización, en la Figura 6.9 se puede ver la localización en este escenario. Se parte de la posición en la que se ha acabado el reconocimiento del mapa, y vemos como el sistema es capaz de seguir la trayectoria que realiza el robot.



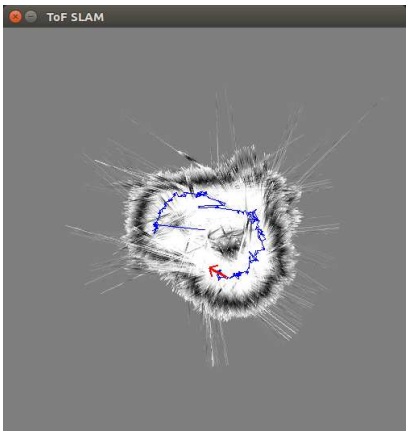
(a) Posición inicial.



(b) Movimiento hacia la izquierda y hacia adelante.



(c) Continua hacia adelante y rotación.



(d) Giro alrededor del obstáculo.

Figura 6.9: Localización del sistema en el escenario 2.

Vemos que el resultado obtenido es correcto, el sistema es capaz de obtener el mapa de este escenario y posicionar correctamente el sistema sobre éste. Sin embargo, durante las pruebas se ha visto que en este escenario, el sistema alguna vez no funcionaba correctamente, debido a que el espacio entre obstáculos y paredes del mapa era demasiado estrecho en comparación con el parámetro σ, y al pasar entre ambos el sistema se perdía. En el siguiente *link* se puede ver el funcionamiento del sistema.

`https://www.youtube.com/watch?v=efHvPh179uQ`

6.2.3 *Escenario 3: Dinámico*

Este escenario es igual que el escenario 2, pero no tiene obstáculos fijos, sino que estos se van a añadir una vez que ya se tenga el mapa e irán desapareciendo. Los parámetros del algoritmo son los mismos que se han utilizado previamente. Mediante el reconocimiento del escenario, se obtiene el mapa de este, mostrado en la Figura 6.10.

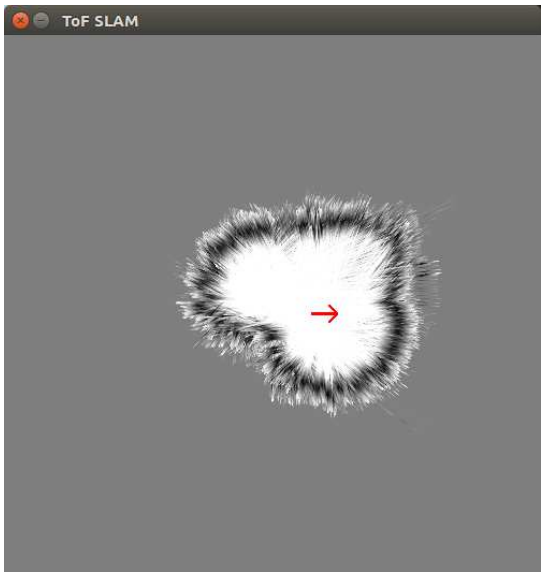


Figura 6.10: Mapa obtenido del escenario 3.

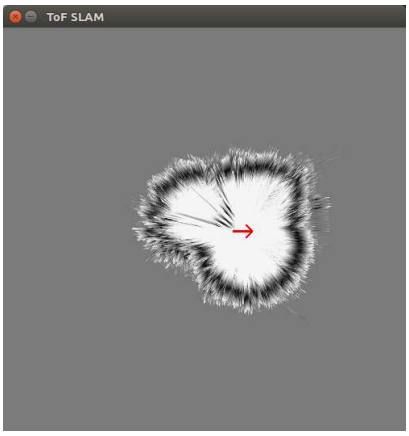
Una vez que se tiene el mapa, se van a incluir los obstáculos mostrados en la Figura 6.11. Primero se incluirá el obstáculo (a), posteriormente se retirará, y se incluirá el obstáculo (b), para posteriormente ser retirado.

(a) Primer obstáculo.

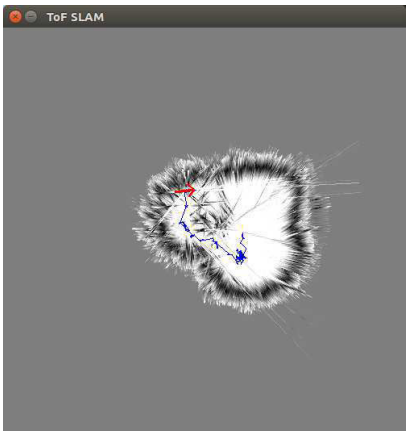
(b) Segundo obstáculo.

Figura 6.11: Obstáculos introducidos dinámicamente.

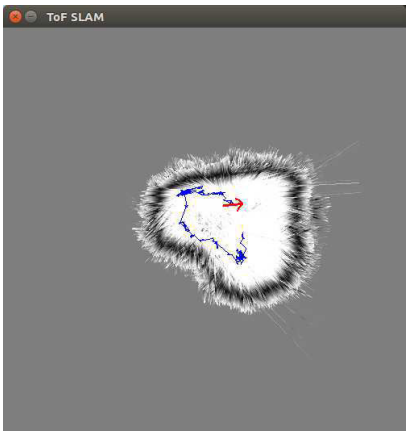
En la Figura 6.12, se ve cómo se detecta el nuevo obstáculo. Tras moverse el sistema por sus inmediaciones éste queda incluido en el mapa y vemos como el sistema puede moverse por sus alrededores localizándose correctamente. Una vez que los obstáculos se eliminan, tras varias pasadas por donde éstos se encontraban quedan eliminados del mapa.



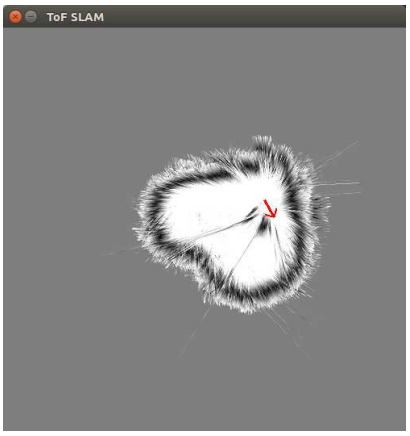
(a) Detección de obstáculo 1.



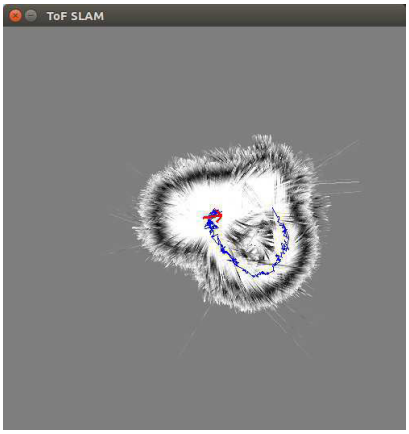
(b) Movimiento alrededor del obstáculo 1.



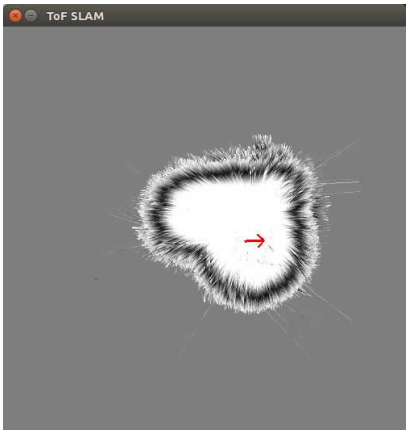
(c) Eliminación del obstáculo 1.



(d) Detección del obstáculo 2.



(e) Movimiento alrededor del obstáculo 2.



(f) Eliminación del obstáculo 2.

Figura 6.12: Detección y localización al añadir obstáculos dinámicos.

Como puede verse, el sistema funciona correctamente con los eventos dinámicos. Estos son detectados en pocos segundos, al igual que

eliminados. No obstante, como solo se dispone de 8 láseres repartidos en 360°, es necesario realizar rotaciones para añadir o eliminar la totalidad del obstáculo en el mapa. En el siguiente *link* se puede ver un vídeo del sistema en funcionamiento en este escenario. En él es posible apreciar mejor como funciona el sistema respondiendo a eventos dinámicos en el mapa.

https://www.youtube.com/watch?v=hwEK3sC6dpY

6.2.4 *Escenario 4: Complejo*

En este escenario se va a recrear una situación más compleja para el algoritmo SLAM. En los escenarios anteriores se tenía un área delimitada y bien definida. Esto hacía que el sistema siempre tuviera medidas de obstáculos. Sin embargo, en este nuevo escenario se va a simular un entorno semiabierto, en el que puede que algunos de los sensores láser no esté detectando un obstáculo, lo que reduce la información que se proporciona al algoritmo. En la Figura 6.13 se puede ver el escenario. El valor de σ para este escenario se ha seleccionado a 40 *cm*.

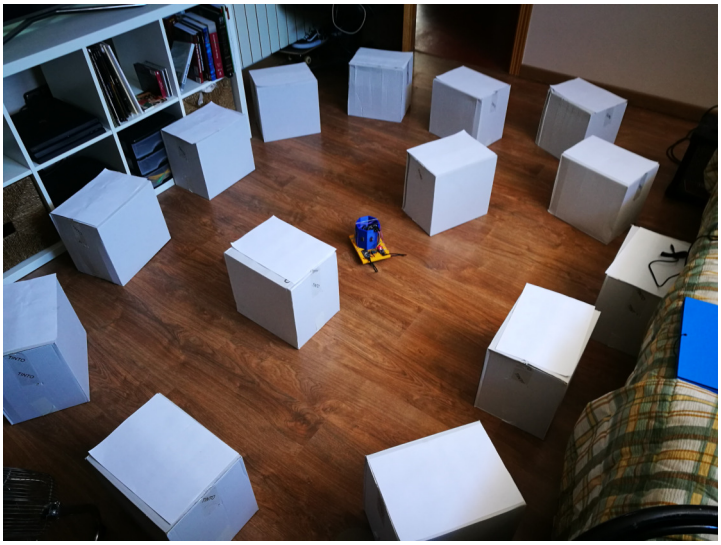
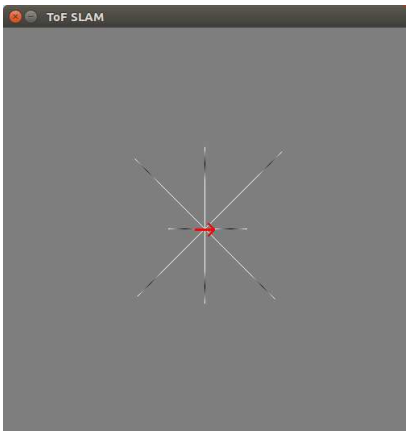
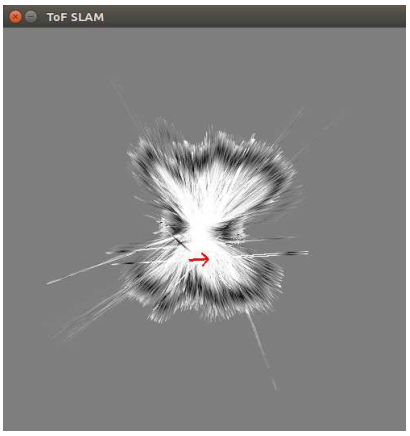


Figura 6.13: Escenario 4.

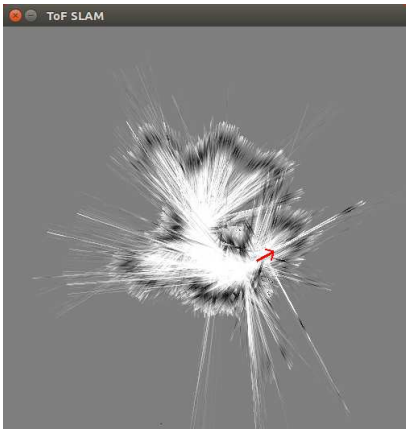
El algoritmo tiene ahora menos referencias con las que estimar la posición. El proceso de reconocimiento será más complejo, ya que cuando el sistema rota sobre sí mismo, en ocasiones pierde las referencias que tenía. En la Figura 6.14 se puede ver diferentes etapas de este proceso.



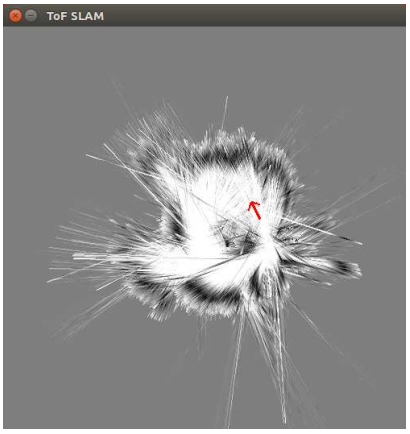
(a) Estado inicial.



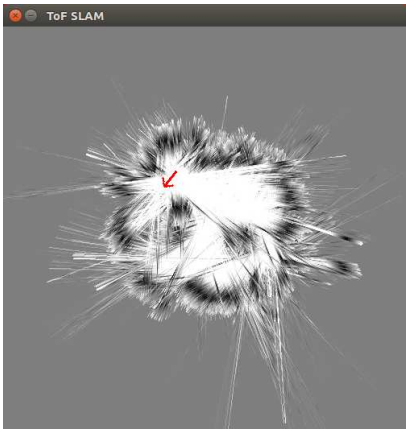
(b) Tras varias rotaciones y movimiento lateral.



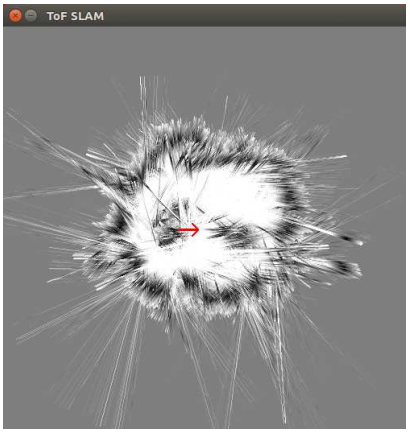
(c) Se avanza y se pasa tras dos obstáculos.



(d) Tras varias rotaciones entre obstáculos, se sale de esa zona.



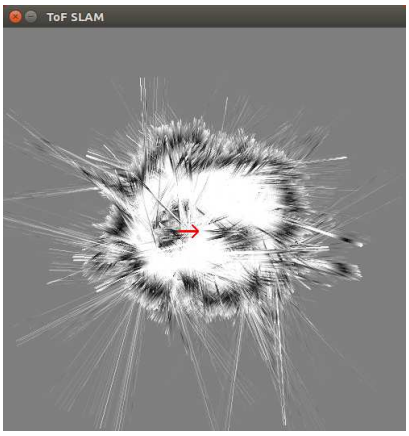
(e) Avanza hasta la zona del mapa aún no reconocida.



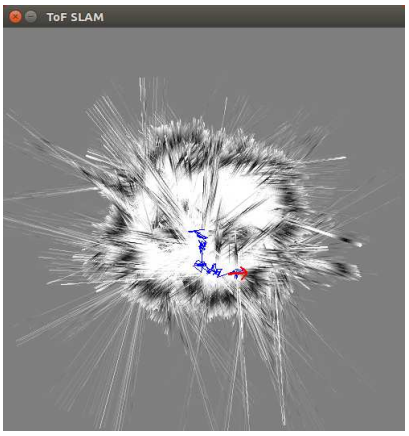
(f) Estado final.

Figura 6.14: Reconocimiento del entorno en el escenario 4.

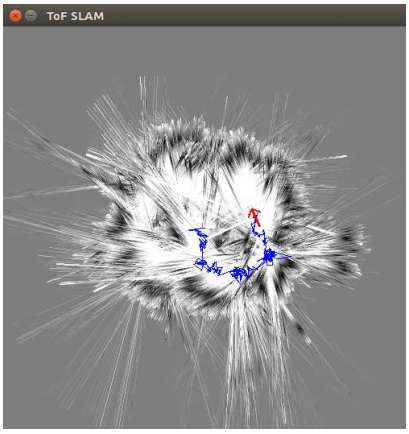
Se obtiene un mapa del entorno y ahora analizamos el proceso de localización, en la Figura 6.15.



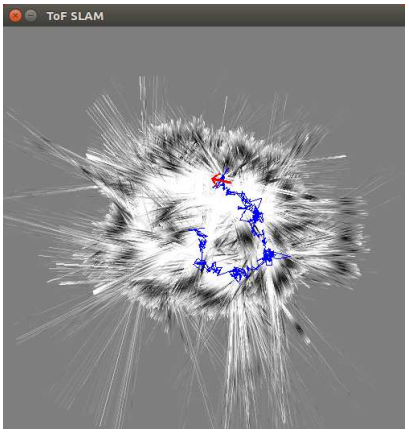
(a) Estado inicial.



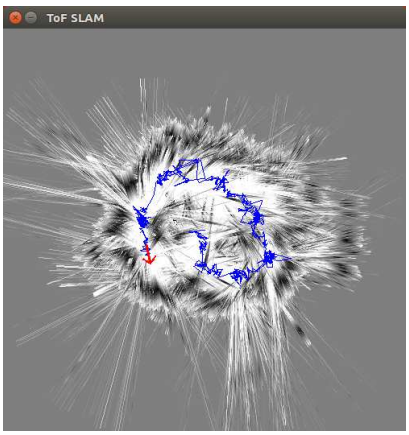
(b) Se avanza hacia la zona de obstáculos del mapa.



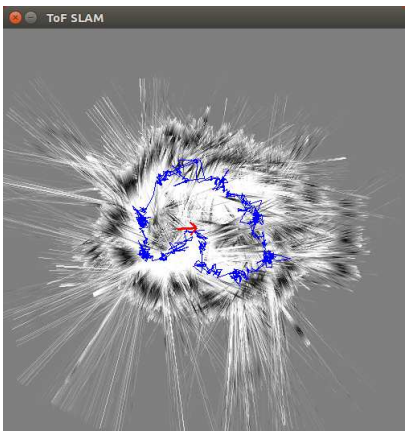
(c) Se pasa tras la zona de obstáculos.



(d) Movimiento alrededor del escenario.



(e) Vuelta a los obstáculos centrales.



(f) Vuelta al punto inicial.

Figura 6.15: Localización del sistema en el escenario 4.

En el siguiente *link* se puede ver un vídeo de todo el proceso. Como vemos el sistema puede obtener un mapa del entorno y localizarse en él, sin embargo, hay que destacar que durante las las pruebas se han registrado varios intentos fallidos en este escenario. Esto es normal, debido a la limitada cantidad de información del entorno que proporciona nuestro sistema por conjunto de datos (ocho datos de láser únicamente).

https://www.youtube.com/watch?v=Xc8vmWCbEC0

6.2.5 *Escenario 5: Secuestro*

En este último escenario se va a analizar cómo responde el sistema al secuestro. Esto consiste en tomar el sistema y moverlo repentinamente a otro lugar del mapa, y ver si éste es capaz de volver a localizarse correctamente. El entorno que se va a utilizar es igual al de los escenarios dos y tres, sin ningún obstáculo. El valor de σ utilizado es de 30 cm.

Tras realizar varias pruebas, vemos que es necesario aumentar la varianza del filtro de partículas, ya que la que estamos utilizando actualmente, al estar más concentrada en el sistema, no proporciona partículas extendidas por el escenario y por lo tanto tras el secuestro el sistema no es capaz de localizarse. En la Figura 6.16, se puede ver la dispersión de las partículas en este escenario.

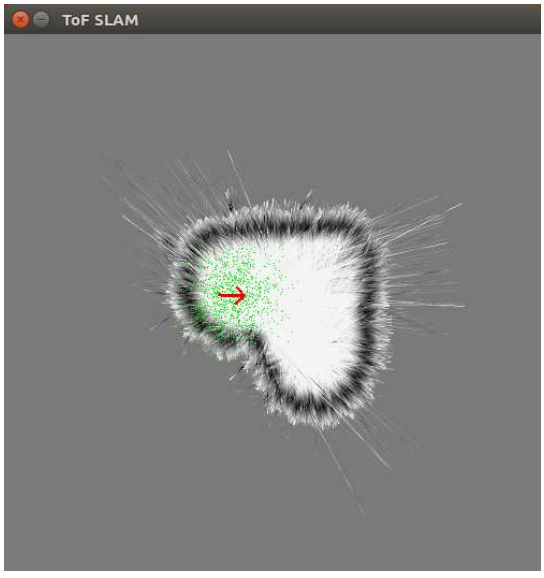


Figura 6.16: Dispersión de las partículas para el escenario 5.

Tras realizar un reconocimiento del entorno, se obtiene el mapa y se sitúa el sistema en la posición que se ve en la Figura 6.17.

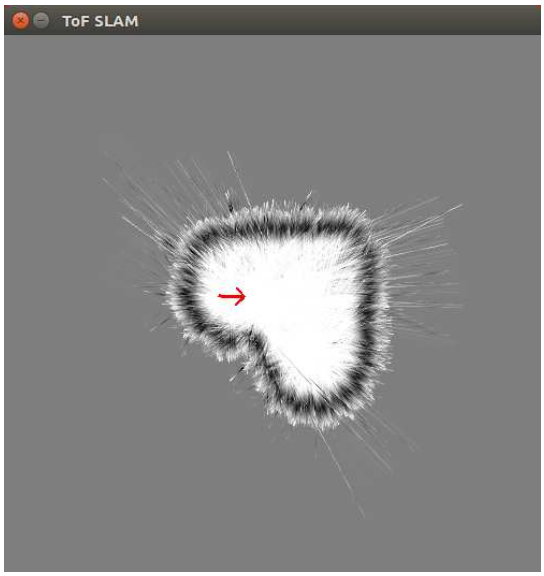


Figura 6.17: Posición inicial del sistema en el escenario 5.

Se coge el sistema y se mueve, levantándolo para que no tenga la referencia de las paredes, hasta la otra esquina redondeada del mapa. Como se puede ver en la Figura 6.18, el sistema es capaz de localizarse, tras haberse perdido, como se puede apreciar en el mapa resultante.

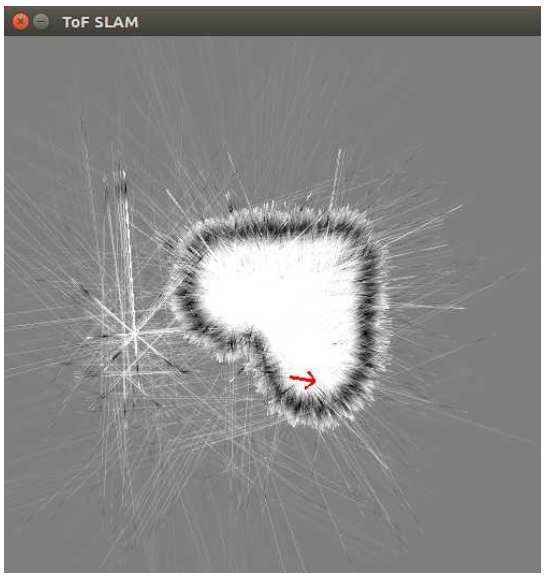


Figura 6.18: Posición final del sistema en el escenario 5.

Hay que realizar varios apuntes a las pruebas llevadas a cabo en este escenario. A pesar de que el sistema es capaz de localizarse, tener una mayor varianza en el filtro de partículas hace que éste, durante

el proceso de reconocimiento, se pierda más veces y le cueste más llegar a converger. Además, hay que tener en cuenta que éste es un escenario complicado, pero no tanto como el escenario cuatro, en el cual, utilizar una varianza del filtro de partículas mayor supondría el incorrecto funcionamiento del sistema.

En el siguiente *link* puede verse el funcionamiento del sistema cuando éste se secuestra y se mueve repentinamente a otra posición del mapa.

https://www.youtube.com/watch?v=_Gs5JebwNTY

7

CONCLUSIONES Y LINEAS FUTURAS

En este capítulo se van a plasmar las conclusiones que se han obtenido del trabajo, así como las líneas futuras que se pueden seguir para ampliar/mejorar el sistema.

7.1 CONCLUSIONES

Se ha implementado un sistema que obtiene datos de ocho sensores láser, dispuestos alrededor de una plataforma octogonal, y de un sensor MARG a la frecuencia necesaria para aplicar posteriormente el algoritmo SLAM. El sistema también procesa los datos obtenidos por los dos tipos de sensores, aplicando a los datos láser un filtro de Kalman para realizar un "suavizado" de los datos obtenidos y eliminar el ruido, y aplicando a los datos proporcionados por el sensor MARG (campo magnético, velocidad de giro y aceleración) un algoritmo de fusión de datos para obtener el ángulo que indica la orientación del sistema.

En esta implementación se ha llevado a cabo un análisis de los sensores utilizados, sus diferentes configuraciones y modos de funcionamiento, con el fin de elegir el que mejor se adapta a los objetivos.

Se ha programado un microcontrolador Arduino, que se encuentra en comunicación los sensores mediante un bus I2C. Este microcontrolador era el encargado de configurar los sensores y obtener los datos de éstos a la frecuencia determinada.

El anterior microcontrolador se ha conectado con un ESP 8266 mediante un bus SPI, creando una estructura de datos que intercambian, para que este último obtenga los datos de los sensores, para posteriormente procesarlos. Se han programado, con la ayuda de bibliotecas externas, el filtro de Kalman aplicado a los datos procedentes de los sensores láser, y el algoritmo de fusión de datos, implementado con un filtro de Madgwick, a los datos del sensor MARG. Mediante una red WiFi, los datos ya procesados se envían a un ordenador personal para aplicar el algoritmo SLAM.

Todo esto se encuentra montado en una placa *protoboard* alimentada por una batería portátil, lo que dota al sistema de movilidad total y ausencia de cables de conexión con el ordenador personal, lo que hace la realización de las pruebas sea mucho más fácil.

El sistema de adquisición es capaz de trabajar a la frecuencia deseada, obteniendo y procesando datos del sensor MARG a 100 *Hz* y datos de los sensores láser a 33 *Hz*, y enviando un conjunto de datos nuevos a través de la conexión WiFi al algoritmo SLAM cada 33 *ms*, como se deseaba.

Se implementa un algoritmo para realizar SLAM con los datos de los sensores láser y la orientación del sistema obtenida a partir del sensor MARG. Este algoritmo corre sobre un ordenador personal.

Para implementar este algoritmo, se ha utilizado como base en un algoritmo SLAM de código abierto que utiliza datos de sensores láser. Tras analizar en profundidad el algoritmo, se ha adaptado para operar con la plataforma desarrollada.

El algoritmo obtiene correctamente los datos a través de la red WiFi, con una frecuencia de 33*Hz*, como se había diseñado. Tras obtener los datos, aplica el algoritmo para posicionarlos con respecto al robot y realizar la estimación de la posición con el mapa que tiene. Tras esto actualiza el mapa y guarda los resultados en un fichero.

Con el fin de evaluar el algoritmo y poder ver claramente sus resultados, se ha creado un *script* en Python, con la ayuda de la biblioteca de desarrollo de videojuegos *Pygame*, que permite ver en tiempo real el mapa que el algoritmo va construyendo, así como la posición actual del sistema y su trayectoria seguida.

Las pruebas realizadas nos permitirán evaluar el desempeño del sistema creado en funcionamiento. Previo a las pruebas es necesario ajustar ciertos parámetros del algoritmo, por lo que se ha realizado un estudio de éstos para entender como afectan a los resultados proporcionados.

Tras el estudio se configuran los parámetros, con los cambios necesarios en cada escenario analizado. Se evalúa el algoritmo en cinco escenarios diferentes, con diferentes dificultades, obstáculos, etc.

El escenario número uno es un rectángulo simple con una esquina redondeada. El sistema es capaz de obtener un mapa del entorno y localizar el sistema mientras se mueve por éste sin ningún problema. Incluso sigue proporcionando una localización correcta aunque se someta el sistema a movimientos muy bruscos.

El segundo escenario tiene mayor complejidad que el primero, ya que su forma no es rectangular. Además, se han añadido dos obstáculos, uno de ellos de tamaño normal, pero otro de ellos más pequeño que la incertidumbre del sistema. El sistema es capaz de obtener el mapa correctamente, aunque sí que vemos que en la zona del obstáculo pequeño, el mapa se deforma, ya que el obstáculo es más pequeño

que el parámetro σ. En cuanto a la localización, el sistema puede localizarse sobre el mapa, pero con mayor dificultad que el escenario anterior, y varias adquisiciones de este escenario han sido erróneas.

El tercer escenario es similar al segundo escenario, pero los obstáculos son más grandes y dinámicos. El sistema es capaz de obtener un mapa sin ningún problema. Al añadir los obstáculos, éstos se añaden al mapa sin afectar a la localización del sistema, y es posible movernos por sus alrededores sin que éste pierda su posición. No obstante, al tener los ángulos una posición fija es necesario realizar ciertas rotaciones para detectar cuánto se expanden estos nuevos obstáculos. La eliminación de los obstáculos del mapa se lleva a cabo sin ningún problema, pero como se comentaba, es necesario que el sistema realice varias rotaciones para eliminarlo completamente.

El cuarto escenario es el más complejo, ya que no se trata de un entorno cerrado completamente como los escenarios anteriores. Ajustando el parámetro σ, el sistema es capaz de obtener un mapa del entorno correcto, aunque con menor precisión que en los escenarios anteriores, y localizarse correctamente en éste.

Finalmente, en el quinto escenario se ha llevado a cabo una prueba de secuestro, es decir, cómo responde nuestro sistema ante un cambio repentino en la posición. El escenario utilizado es similar al de los escenarios dos y tres, pero sin obstáculos. Para conseguir un correcto funcionamiento en este escenario, se ha tenido que aumentar la varianza del filtro de partículas. Esto permite que el sistema obtenga la posición correcta tras el secuestro, sin embargo, aumentar la varianza tiene otros problemas asociados, como mayor inestabilidad del sistema, que en escenarios complejos pueden suponer el incorrecto funcionamiento del sistema.

Tras analizar los resultados las conclusiones obtenidas se muestran a continuación:

- El sistema es capaz de obtener un mapa y localizarse correctamente en entornos ortogonales, siendo capaz de obtener un mapa de éste y localizarse en él sin ningún problema.
- Los mapas obtenidos cuando en el escenario se encuentran obstáculos del orden o más grandes que el parámetro ancho de función de probabilidad del obstáculo, σ, son correctos, y la localización proporcionada también lo es.
- Se cometen ciertos errores cuando se encuentran presentes obstáculos más pequeños que el parámetro σ. Esto se debe a que la representación del obstáculo en el mapa es mucho más grande de la realidad y esto deforma demasiado el mapa.

- Existen errores cuando el sistema pasa por zonas muy estrechas (también relacionado con lo anterior), haciendo que en ocasiones el sistema se pierda en el mapa.
- La resolución del sistema no es muy grande debido a que en cada conjunto de datos solo hay ocho medidas separadas 45° cada una.
- Es necesario realizar continuamente rotaciones sobre él mismo para poder estimar las inmediaciones de éste correctamente, y así poder obtener un mapa fiel a la realidad y una posición correcta. Esto es debido al punto anterior.
- El sistema es muy dependiente del número de láseres que cuentan con detecciones. Esto es, cuando un láser no detecta un obstáculo, su peso en el mapa final no cuenta lo mismo, por lo que si hay varios láseres sin detecciones, el algoritmo lo tendrá más difícil.
- Ajustando los parámetros, es posible que el sistema se recupere de secuestros, con las contrapartidas comentadas.

Como conclusión final, queda demostrado que el sistema implementado es capaz de obtener el mapa de los entornos en los que ha sido probado, así como localizarse correctamente. Todo esto únicamente con información del entorno proporcionada por únicamente ocho sensores láser, sin ningún tipo de rotación automática, y sin información de odometría.

7.2 LÍNEAS FUTURAS

Tras todas las pruebas llevadas a cabo con el sistema, se han podido encontrar los puntos débiles que éste presenta. Las posibles mejoras que permitirían un mejor funcionamiento del sistema y sería interesante analizar los resultados obtenidos con estas se exponen a continuación como futuras líneas:

- Una de las principales continuaciones a este trabajo, con mayor impacto en los resultados, sería incorporar un movimiento mecánico de los sensores, para así poder obtener por cada conjunto de datos más de los ocho rayos que se obtienen actualmente. Esto permitiría que el algoritmo trabajara con un mayor número de referencias, obteniendo unos mejores resultados.
- Estudiar el impacto de la función de distribución con la que se generan las diferentes partículas del filtro. Éste es el encargado de estimar la nueva posición mediante la posición de cada una de las partículas, por lo que esto puede tener un impacto importante en la estimación de la posición.

- Con el fin de abordar el problema del secuestro sin perder la estabilidad que presenta el sistema, debido a utilizar una varianza para el filtro de partículas no muy grande, podría implementarse una función que detectará grandes variaciones entre el mapa y los valores obtenidos actualmente. Cuando esto se detectara, modificaríamos la varianza del filtro de partículas por una mayor, para así poder recuperar la posición correcta del sistema. Esto es algo similar al cierre del lazo.

- Sería interesante comprobar los resultados obtenidos realizando SLAM en las tres dimensiones en un entorno sencillo. Para esto, sería necesario añadir algún sensor para obtener la altura que se tiene. En cuanto a la actitud del sistema, no haría falta añadir nada más ya que se dispone de los tres ángulos de navegación, *roll* , Φ, *pitch*, θ, y *yaw*, Ψ.

- Con el fin de montar el sistema sobre la aplicación para la que ha sido pensada, un dron, se podría tratar de aligerar la plataforma y tratar de optimizar el consumo del sistema para garantizar una mayor duración en operación.

Parte III

BIBLIOGRAFÍA

BIBLIOGRAFÍA

[1] H. Shoudong y D. Gamini, "Robot localization: An introduction", en *Wiley Encyclopedia of Electrical and Electronics Engineering*. 2016, págs. 1-10.

[2] J. J. Leonard y H. F. Durrant-Whyte, "Mobile robot localization by tracking geometric beacons", *IEEE Transactions on Robotics and Automation*, vol. 7, n.º 3, págs. 376-382, 1991, ISSN: 1042-296X.

[3] R. Smith, M. Self y P. Cheeseman, "Estimating uncertain spatial relationships in robotics", en *Proceedings. 1987 IEEE International Conference on Robotics and Automation*, vol. 4, 1987, págs. 850-850.

[4] M. R. B. Soren Riisgaard, *SLAM for Dummies, A Tutorial Approach to Simultaneous Localization and Mapping*. 2005.

[5] *Accelerometer and gyroscopes sensors: Operation, sensing and applications*, 5830, Maxim Integrated, 2014.

[6] J. F. Vasconcelos, G. Elkaim, C. Silvestre, P. Oliveira y B. Cardeira, "Geometric approach to strapdown magnetometer calibration in sensor frame", *IEEE Transactions on Aerospace and Electronic Systems*, vol. 47, n.º 2, págs. 1293-1306, 2011.

[7] R. Barry, *Mastering the FreeRTOS Real Time Kernel, A Hands-On Tutorial Guide*. 2016.

[8] R. E. Kalman, "A new approach to linear filtering and prediction problems", *Journal of Basic Engineering*, 1960.

[9] A. P. A. Mohinder S. Grewal, *Kalman Filterring: Theory and Practice*. 2001.

[10] R. Faragher, "Understanding the basis of the kalman filter via a simple and intuitive derivation [lecture notes]", *IEEE Signal Processing Magazine*, vol. 29, n.º 5, págs. 128-132, 2012.

[11] M. Euston, P. Coote, R. Mahony, J. Kim y T. Hamel, "A complementary filter for attitude estimation of a fixed-wing uav", en *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, págs. 340-345.

[12] S. O. H. Madgwick, A. J. L. Harrison y R. Vaidyanathan, "Estimation of imu and marg orientation using a gradient descent algorithm", en *2011 IEEE International Conference on Rehabilitation Robotics*, 2011, págs. 1-7.

[13] S. O. Madgwick, *An efficient orientation filter for inertial and inertial/magnetic sensor arrays*. 2010.

[14] H. Durrant-Whyte y T. Bailey, "Simultaneous localization and mapping: Part i", *IEEE Robotics Automation Magazine*, vol. 13, n.º 2, págs. 99-110, 2006.

[15] T. Bailey y H. Durrant-Whyte, "Simultaneous localization and mapping (slam): Part ii", *IEEE Robotics Automation Magazine*, vol. 13, n.º 3, págs. 108-117, 2006.

[16] M. Montemerlo, S. Thrun, D. Koller y B. Wegbreit, "Fastslam: A factored solution to the simultaneous localization and mapping problem", en *In Proceedings of the AAAI National Conference on Artificial Intelligence*, AAAI, 2002, págs. 593-598.

[17] B. Steux y O. E. Hamzaoui, "Tinyslam: A slam algorithm in less than 200 lines c-language program", en *2010 11th International Conference on Control Automation Robotics Vision*, 2010, págs. 1975-1979.

[18] J. E. Bresenham, "Algorithm for computer control of a digital plotter", *IBM Systems Journal*, vol. 4, n.º 1, 1965.

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá